T S X - P L U S

OVERVIEW

TSX-Plus is a high performance, general purpose, time sharing operating system for Digital Equipment Corporation PDP-11 and LSI-11 computers. TSX-Plus provides the functionality of the DEC RT-11 operating system up to 20 concurrent time-sharing users.

SYSTEM PERFORMANCE IMPROVEMENTS

TSX-Plus overlaps terminal interaction time, I/O wait time and CPU compute time for all jobs on the system. The result is tremendous increase in the productivity of the computer. Compute-bound jobs can soak up idle time while many users perform file editing, program development or file inquiry. TSX-Plus allows the same machine to simultaneously support production use and program development.

RT-11 COMPATIBILITY

TSX-Plus was designed from the ground up to efficiently support a wide variety of RT-11 programs. TSX-Plus contains no RT-11 emulator as some other time-sharing systems do, rather, TSX-Plus supports RT-11 system service calls as its basic mode of operation. The result is lower system overhead and substantially improved performance over systems that must emulate RT-11 services. Most RT-11 programs can be used with TSX-Plus without change or even having to be relinked. TSX-Plus interfaces with standard RT-11 device handlers and supports RT-11 utility programs such as PIP, DIR, DUP. The TSX-Plus keyboard commands are an extended set of those provided by RT-11. TSX-Plus fully supports CCL commands such as COMPILE and EXECUTE.

GENERAL PURPOSE SYSTEM

TSX-Plus is truly a general purpose time-sharing system. It can simultaneously support a wide variety of jobs and programming languages including COBOL-Plus, FORTRAN, BASIC, DIBOL, PASCAL, MACRO, TECO and KED. TSX-Plus in use in educational business, scientific and industial environments. It can concurrently support commercial users doing transaction processing engineering users performing scientific processing, and system programmers doing program development.

MEMORY MANAGEMENT

TSX-Plus fully utilizes the memory management facilities of the PDP-11 computer. This allows multiple jobs to be in memory at the same time and allows rapid switching between executing jobs. A "MEMORY" command can be used to control the memory space allocated to a job. Up to 56Kb of memory is available to each job.
When there are more jobs active on the system than total memory space, TSX-Plus uses a priority and time-slice scheduling system to determine which jobs to keep in memory and which to swap out to a disk file. The memory management facility also protects the system by preventing user jobs from halting the machine or storing outside their program regions.

SYSTEM ADMINISTRATIVE CONTROL

TSX-Plus supports an optional logon facility that requires users to enter a project-programmer number and password before they can access the system. The logon facility also keeps track of system usage statistics including the number of logons, total connect time and total CPU usage of each account.

PRINTER SPOOLING SYSTEM

TSX-Plus provides a convenient and powerful facility for spooling output to line printers and other output devices. The spooling is automatic. If the line printer is spooled, directing output to device "LP" from a program causes the output to be spooled. A print file may designate the name of the form on which it is to be printed. When a form change is required, the spooled device is suspended and a message is sent to the operator requesting the required form. After mounting the form the operator may cause the spooling system to print a form alignment file. Once a form is mounting, all files requiring the form are printed without further operator intervention. The operator may also lock a particular form on the printer, thus preventing automatic form change requests.
Other commands allow the operator to list the names of all pending print requests and required form names, delete files, and reprint a portion of a print file. The TSX-Plus spooler will simultaneously drive any number of spooled devices.

SHARED FILE RECORD LOCKING

Several cooperating programs that wish to update a common file can coordinate their file access by using the TSX-Plus file enqueue/dequeue facility. A program may request exclusive access to one or more blocks within a file. If the blocks are not enqueued by another program, the access is permitted. Otherwise, at the program's option, an error code is returned or the program is suspended until the desired blocks are available.

INTER-PROGRAM MESSAGE COMMUNICATION

TSX-Plus offers a message communication facility that allows running programs to send messages to each other. Messages are transmitted through named "message channels". A program can queue messages on one or more message channels. Receiving programs can test for the presence of messages on a named channel and can suspend their execution until a message arrives. A message can also be queued for a program that will run at some time later.

PROGRAM DEBUGGING FACILITY

If the "/DEBUG" switch is specified with the RUN command, the program being started is executed under control of the TSODT debugging program. TSODT allows the program being run to be examined or modified and allows breakpoints to be set. This debugging facility works by loading a relocatable copy of the debugging program into the top of memory. The program being debugged does not have to be modified or even relinked.

PERFORMANCE ANALYSIS FACILITY

TSX-Plus includes a performance analysis facility that can be used to monitor the execution of a program and determine what percentage of the run time is spent at various locations within the program. When the performance analysis facility is being used TSX-Plus examines the program being monitored when each clock tick occurs (50 or 60 times per second) and notes at what location in the program execution is taking place. Once the analysis is completed the TSX-Plus performance reporting program can be used to produce a histogram showing the percentage of time spent at various locations during the monitored run.

VIRTUAL LINES AND DETACHED JOBS

TSX-Plus provides a facility known as "virtual lines" that allows one time-sharing user to control several simultaneously running programs from a single terminal. This is done by allowing the user to logically disconnect the terminal from the original time-sharing line and reconnect it to a different "virtual" line.
When a program that is not currently connected to a time-sharing line writes output to the terminal, the output is stored in a buffer area: when the buffer is filled the program is suspended until the user reattaches to the program and accepts the queued output.
Switching to a virtual line is accomplished by typing control-W followed by a digit that selects the line to which the user wishes to attach. Virtual lines are useful in situations in which it is desirable to run a lone "number crunching" job without having to tie up a terminal.
Another TSX-Plus feature is "Detached jobs". A detached job is a job that is executing but not connected to any time-sharing line. Communications with other jobs is accomplished by use of the inter-job message communication facility.

POWERFUL COMMAND FILES

TSX-Plus command files allow up to 10 parameters to be specified on the command file command line. These parameters may be substituted into the command file at arbitrary points. It is possible with TSX-Plus command files to have all program terminal input come from the command file-including input accepted by use of the TTYIN EMT. Command files that have names that do not conflict with standard system commands may be placed on the system device and executed as system keyboard commands without having to specify the leading at-sign. This provides a facility that allows the user to extend the set of system commands.

LOCKED PROGRAMS

A program can be associated with a time-sharing line or a particular project-programmer number so that the program is automatically stared when the user logs on. The program can be locked to the user so that he cannot escape from the program.
This is useful in secure environments where some users are only allowed to run specific application programs.

EXECUTION SCHEDULING

Three execution priorities and three time-slice values are used to schedule jobs. Users who have just typed a line of input are given the highest priority. Such a user will interrupt any other currently executing user when that user exceeds a short time-slice (typically less than 1 second). If the interactive user executes longer than the short time-slice the user is classified as compute bound. Compute bound users run at a lower priority than interactive users, but get a longer time-slice. Compute bound users compete for the CPU on a round-robin basis. The lowest execution priority is given to compute bound tasks running on virtual time-sharing lines that are not currently connected to time-sharing terminals. These low priority tasks execute only when no higher priority task needs service. This scheduling algorithm gives fast response to interactive users but minimizes job swapping overhead.

HARDWARE REQUIREMENTS

TSX-Plus will run on any PDP-11 or LSI-11 computer that has memory management facilities and at least 96Kb of memory. The system must also have a disc suitable for program swapping (the swapping disc can be used for regular file storage as well). Time-sharing lines can be connected to the system through DL-11 or DZ-11 communication devices. Both hard-wired and dual-up lines are supported by TSX-Plus.
A DEC RT-11 license is required as RT-11 handlers and utility programs are used with TSX-Plus.

TSX-Plus
Operating System

Phil Sherrod
S&H Computer Systems, Inc.
Nashville, Tennessee

ABSTRACT

TSX-Plus is a high-performance, general-purpose time-sharing operating system for PDP-11 and LSI-11 computers. The system service calls (EMT's) provided by TSX-Plus are compatible with those offered by the RT-11 operating system. With few exceptions programs that run under RT-11 can be used with TSX-Plus without change or even relinking. The keyboard commands are also compatible with RT-11 and most RT-11 users find that they can readily adapt to TSX-Plus without difficulty. Features provided beyond those of RT-11 include a transparent line-printer spooling system, a logon and usage accounting system, shared file access control, inter-job message communication, a program performance monitoring facility and command files with parameters. Up to 20 concurrent users can be supported by TSX-Plus.

History of TSX-Plus

TSX-Plus is an outgrowth of the TSX (Time-Sharing Extension) facility that was developed over a period of years beginning in 1975. Although TSX and TSX-Plus provide similar facilities (RT-11 compatible time-sharing) they are fundamentally different in internal organization. TSX is not a separate operating system but rather a program that runs in conjunction with the RT-11 operating system to allow multiple users to time-share the facilities of RT-11. TSX acts as an interface between multiple user programs and terminals but calls on RT-11 to perform low-level system service functions. Although TSX is quite successful and popular (it is in use at over 1,000 sites) it has a number of limitations. Since TSX and RT-11 both had to be in memory with the user program, the memory space available to the time-sharing user was typically restricted to about 32Kb. Also TSX did not require (or use) memory management facilities so only a single program was held in memory at a time; this resulted in heavy program swapping if many users were accessing the system.

The TSX-Plus project was begun in 1979 with the general goal of developing a much higher performance system and removing the memory size limitations. In order to achieve these goals, two fundamental design decisions were made early in the project: First, in order to allow multiple users to be kept in memory and to provide a program region of up to 56Kb for each user, the 11/23 and 11/34 type memory management facilities were required. Secondly, it was decided to make TSX-Plus into a self-contained operating system which would not run in conjunction with RT-11 but which would rather replace it. These decisions necessitated a major redesign and reimplementation of the system. Facilities such as I/O queueing and device handler interfacing as well as file directory operations that had previously been provided by RT-11 running under TSX had to be implemented as native mode services in TSX-Plus. Fundamental changes were also required in areas such as memory management and job execution scheduling. The major parts of TSX that were used more-or-less intact in TSX-Plus were the terminal driver and the keyboard command interpreter.

General System Operation

The TSX-Plus system that has resulted from this development effort is quite similar to TSX from a functional point of view (although it provides

much higher-performance) but is radically different internally. TSX-Plus runs as a stand-alone operating system replacing RT-11. An RT-11 license is still required, however, because RT-11 device handlers are used with TSX-Plus as well as utility programs such as PIP, DUP, DIR, LINK, MACRO, etc. Each TSX-Plus time-sharing job can access up to 56Kb of memory and as many time-sharing jobs are held in memory as will fit. Switching between in-memory jobs is extremely fast since it involves only performing a context change which includes reloading the memory management registers. If there are more active users than will fit in memory, it swaps jobs to a disk file on an execution-priority and time-slice basis. I/O wait time for user jobs and system swapping is fully overlapped with execution of other jobs.

The time-sharing facilities provided by TSX-Plus are general purpose and are suitable for concurrent program development and production use. Programs supported by TSX-Plus include MACRO, FORTRAN, BASIC, COBOL-Plus, DIBOL, DBL, RTSORT, APL, PASCAL, LINK, TECO, KED, EDIT, DIR, PIP, DUP and DUMP.

Since TSX-Plus is not an emulator but rather provides RT-11 compatible system service functions as its native mode, there is very little extra system overhead involved in running a program under TSX-Plus rather than RT-11; total system throughput is of course greatly increased because I/O wait time and execution time of multiple jobs can be overlapped.

TSX-Plus Program Environment

Each TSX-Plus job appears to running under a normal RT-11 system. The virtual memory region available to the job begins at (virtual) address zero and extends upward to a size that can be controlled by the user but which may not exceed 56Kb. The upper 8Kb (virtual) region of each job (160000 through 177777) is mapped to a simulated RMON so that programs that access offset cells in the base of RT-11 will operate properly under TSX-Plus. System service calls such as .TTYIN and .GTLIN that are directed to the console terminal under RT-11 are directed to the time-sharing terminal controlling the job.

The keyboard commands provided by TSX-Plus are compatible with those provided by RT-11. TSX-Plus fully supports CCL commands such as COMPILE and EXECUTE.

The file system accessed by TSX-Plus jobs is compatible with that provided by RT-11. Disks may be transported freely between TSX-Plus and RT-11 systems.

Virtual Lines

TSX-Plus provides a facility known as "virtual lines" that allows a time-sharing user to control several programs from a single terminal. When a user initially logs onto TSX-Plus he is said to be connected to his "primary" line. At any time the user may disconnect from the primary line and connect to a secondary line by typing Control-W followed by a digit that selects which secondary line is to be accessed. The program that was running on the primary line is not lost and its execution is not affected (but it runs at a lower priority as long as it is disconnected from the terminal). If the disconnected program generates terminal output, it is held in the line's terminal output buffer until the terminal is reconnected to the job. If the terminal output buffer is filled, the job is suspended until the output is printed. The number of secondary lines that each user may access is specified when the system is generated. The use of secondary lines allows a user to control several simultaneously running programs without having to wait for them to finish execution. A typical use is to start a long assembly on a secondary line and then proceed with other work on a different logical line while the assembly runs to completion. The bell is rung on the terminal if a disconnected job reaches the point where it is waiting for more terminal input or its terminal output buffer fills.

A related TSX-Plus facility is "Detached Jobs". A detached job is a job that is not associated with any time-sharing terminal. Detached jobs may be started by time-sharing users, running programs or may be specified for automatic startup during system initialization. Detached jobs can only communicate with other jobs by using the TSX-Plus inter-job message communication facility (described below).

TSX-Plus terminal Driver

The TSX-Plus Terminal driver allows the user to specify "activation characters" which will be treated as field delimiters. If, for example, a program wants to handle the rubout character in a special way, it could declare that character to be an activation character. Then whenever the rubout character is received by TSX-Plus, it is not echoed to the terminal and is not acted on by TSX-Plus but rather serves as a field delimiter and is passed to the program as the last character of input for the current field.

It is also possible to specify field size as an activation condition. If this is done, the input field is considered to be terminated when it is filled to

the specified size even if no activation character has been received yet. A related facility allows a field size limit to be declared. If characters are typed beyond the end of the specified field size, they are discarded and the bell is rung rather than echoing them to the terminal. This is useful in form fill-out applications where it is desirable to constrain the size of the field a user can type into.

TSX-Plus uses "deferred" character echoing. This means that if a user types ahead, the character echoing is held until the program is ready to accept the input. This is essential in applications where cursor addressing is being done — the echoing of the typed ahead characters must be deferred until the program has the chance to send the cursor positioning commands to address the next input field where the typed-ahead characters are to be placed.

It is possible under TSX-Plus to associate a completion routine with a user specified terminal "break" character. If this is done, the completion routine will be entered each time the break character is typed at the terminal. This facility could be used to signal a compute-bound program to display its current status or to trigger entry into a program debugging facility (COBOL-Plus does just that).

File Access Control

TSX-Plus provides a file access control facility that can be used to coordinate access to a file that is being shared by several programs. There are two basic aspects to the file access control facility: file opening control and record locking control.

After opening a file (using a .LOOKUP) a program may determine if other programs that already have access to the file are willing to share the file. This is done by executing an EMT which specifies whether the file is being opened for input or update and declares what type of access other users may be granted. Three access classes may be specified: *Exclusive* means that the job demands exclusive access to the file and will allow no other jobs to access it in any fashion; *Protected* means that the job is willing to allow other jobs to read the file but wishes to prohibit anyone else from updating the file; *Shared* means that the program is willing to allow other programs to read and update the file.

After the program has gained access to the file, it can coordinate its access to specific blocks in the file by using the record locking facility. This facility allows a job to lock one or more blocks in the file. Attempts by other jobs to lock the same blocks will be rejected. When a program attempts to lock a block it has the choice of receiving an error code if the block is locked by another job or being suspended until the requested block becomes available.

Another shared file control facility allows a program to ask the system if the shared file has been updated by any other user. A status code is returned by the system indicating whether a write operation has been directed to the file by any other job since the last time the job checked for file modification.

Inter-job Message Communication

TSX-Plus includes an inter-job message communication facility that allows jobs to send messages to each other through named "message channels". The message facility allows a program to perform three basic operations: queue a message on a named channel, test for a pending message on a channel, and wait for a message on a channel. A message channel name (1 to 6 characters) is specified with each message transmitted by a job. This name is arbitrary and is only used as a means to identify the message to other jobs. The transmitted message is stored in system buffer space. This means that the sending job can perform other functions or even exit without disturbing the message it has queued on the channel. A receiving program specifies the name of the channel from which it wishes to draw a message. It may request to be notified if no messages are available on the channel or it may be suspended until a message arrives on the channel. If several messages are queued on the same channel name, they are stored and delivered to the receiving program in the order they were transmitted. A program suspended while waiting for a message is given a very low memory-use priority and is likely to be swapped out of memory until a message arrives.

A good example of the use of the message communication facility and detached jobs is the way COBOL-Plus performs a disk sort. When the system is started, the RTSORT disk sort program is started as a detached job. Given the correct command it will enter "message mode" which causes it to accept sort commands from a named message channel. When a COBOL-Plus program executes a SORT statement, the COBOL-Plus runtime system generates the appropriate RTSORT command and sends it as a message to RTSORT. The COBOL program then waits for a reply message from RTSORT. RTSORT is activated when a message arrives on the channel it is waiting for, performs the sort and then sends a reply message indicating the success or failure of the sort. The COBOL program which is asleep waiting for the

reply is then reactivated when the reply message arrives and continues execution.

TSX-Plus Spooling Facility

TSX-Plus provides a transparent printer spooling system. The term *transparent* means that output is directed to the spooled printer by simply writing to device "LP:" (or whatever name the printer has). It is not necessary for the program to first create a disk file and then queue that file for printing. The TSX-Plus spooling system intercepts output directed to the printer and diverts it to a system managed spool file. This spool file is used for all users and all active spooling operations. If the spool file becomes full, programs writing output to the spooled printers are suspended until space becomes available.

A program may associate a form name with a print file. If this is done TSX-Plus keeps track of which files need which forms and issues form mount commands to the operator when it is ready to print a file that requires a form different from the one that is currently mounted on the printer. Between form changes the operator may print form alignment files. When a new form is mounted, TSX-Plus prints all of the files requiring that form before requesting a new form mount. It is also possible for the operator to lock a particular form mount in which case TSX-Plus will not attempt to print files requiring a different form until the form is unlocked.

Command Files

TSX-Plus command files are very similar to RT-11 command files but provide a couple of extensions. Up to 9 parameter strings may be specified with a TSX-Plus command file. The point at which a particular parameter is to be inserted in the body of a command file is indicated by placing an up-arrow character followed by a digit in the range 1 to 9 in the command file at the point where the parameter insertion is to be made. Parameters may be inserted in any order and may be inserted repeatedly if desired. The parameters for a higher-level command file are pushed on a stack when a lower-level command file is called. Parameters may also be passed down to lower-level command files by using the up-arrow-digit sequence as the argument to the lower-level command file.

It is possible to create user-defined system commands using command files. When the TSX-Plus keyboard command interpreter receives a command, it searches to see if it is a recognized system command (COPY, DIR, R, etc.); if it is not, it then looks for a command file with the same name as the command. If such a command file is found it is started but its listing is suppressed (implicit SET TT QUIET).

It is possible using TSX-Plus command files to direct *all* program terminal input to the command file. This allows command files to be created that contain TECO commands or data for FORTRAN or BASIC programs.

Logon Facility

TSX-Plus includes an optional logon facility that requires users to enter project-programmer numbers and passwords to gain access to the system. If this facility is used, TSX-Plus also performs system usage accounting for each project-programmer number.

An additional facility associated with the logon feature is the ability to restrict which devices and files can be accessed by a given user. A user can be constrained so that he has access to only a specified set of devices and/or files with a particular name or extension. The access may be limited to read-only or read and write.

File Directory Caching

TSX-Plus maintains an in-memory cache of file directory entries to speed up file operations. When a lookup is done on a file, the directory cache is searched first before accessing the directory on the device. If the requested file entry is found in the cache, the lookup can be completed without having to access the device at all. Enter, Delete and Rename requests always update the directory on the accessed device to avoid the possibility of directory corruption in the case of a system crash or power failure.

The directory for the system disk is always cached; directories for other devices are only cached if the device is introduced to the system by use of the "MOUNT" command. If the directory for a device is being cached, the "DISMOUNT" command must be used to tell TSX-Plus that a device is being dismounted before it can be removed or a new pack mounted.

The file directory cache is common to all users and all devices being cached. The entries in the cache are maintained in most-recently-used order and the least-recently-used entry is replaced whenever a lookup is done on a file whose entry is not in the cache.

Performance Monitor

TSX-Plus includes a facility that allows the execution of a program to be analyzed to determine what percent of the run-time is being spent in each region of the program. When a program is run with the performance analysis facility turned on, TSX-Plus samples the program counter for the program on a line-clock frequency basis and develops a table showing the number of clock ticks that occured while the program was executing in each address region. After the program completes its execution, the TSXPM performance analysis reporting program is used to print a histogram showing the percent of execution time spent in each address region of the program. The range of addresses being monitored and the resolution of the address regions are user selectable. The address regions may be as small as 1 word.

Program Debugging Facility

If a program is started by typing "RUN/DEBUG program-name", TSX-Plus executes the program under the control of the TSODT debugging facility. Starting a program in this fashion causes TSX-Plus to load a relocatable copy of the TSODT program into the top of memory above the program being started. TSODT is then entered with the starting address of the program being debugged in register zero. TSODT can be used to set breakpoints and examine or modify the program. This allows a program to be debugged without having to link the debugger with the program.

Job Execution Scheduling

TSX-Plus uses a dynamic priority system for scheduling jobs for execution. Each job in the system is in some known execution state which has an associated execution priority. The transition from one execution state to another is caused by external events such as reception of an activation character or an I/O interrupt or a time-quantum expiring. The actual job scheduling algorithm is simple: it selects the highest-priority job that is in memory and wants to run. The interesting part is the way job state transitions occur.

When an event occurs that allows a job in a wait state to resume execution (such as receiving an activation character or an I/O interrupt), the job is placed in a high-priority execution state. This allows the job to interrupt the execution of some lower-priority job. However, the job is assigned a short time-quantum. If the job does not enter a wait state before the time-quantum expires, its execution state is changed to compute-bound (which has an associated lower execution priority) and the job is requeued at the tail of the list of other compute-bound jobs. Compute-bound jobs are selected for execution if there are no higher-priority jobs. When execution is started for a compute-bound job it is given a moderately long time-quantum (typically several seconds); if a high-priority job comes along before the time-quantum expires, the compute-bound job is suspended and requeued at the tail of the list of compute-bound jobs. If the compute-bound job runs to the end of its time-quantum, it is requeued at the tail of the compute-bound job list and the next compute-bound job is selected for execution. Compute-bound jobs running on virtual lines that are not connected to terminals are given an even lower priority state. They execute only if there are no high- or normal-priority users; however, when they do execute they are given a long time quantum (typically 15 seconds). Naturally this time-quantum can be interrupted if a higher priority job comes along.

This method of scheduling execution has a number of desirable attributes. Interactive users receive snappy response even when there is a heavy load of compute-bound users. Compute-bound jobs are given a longer time-slice so job switching does not occur too often for them. Low priority disconnected jobs soak up idle CPU time but do not interfere with the responsiveness of the system.

If there are more active users running than can be simultaneously accomodated in memory, it is necessary to swap jobs to a disk file to share the memory resource. The algorithm used to select jobs for swapping is very similar to that used to schedule jobs for execution. It too is based on the dynamic job execution priority. Each time an execution state change occurs the job swapper is called to see if a swap is appropriate. It scans the list of jobs in the order of high priority to low priority. If it finds a job that wants to run but is not in memory it starts a backward scan up the job list from lowest priority toward higher priority. If it finds a lower-priority job that is in memory it selects that job to be outswapped. If there is no lower priority job then no swap occurs.

Future Directions

A number of extensions are being considered for inclusion in future versions of TSX-Plus. The first is likely to be a real-time support package. The basic functions to be provided are to allow a program to connect interrupts to completion routines, allow a program to access the I/O page, allow a program to lock itself in memory and allow a program to set its execution priority.

The second major extension under consideration is a fundamental redesign of the file directory structure. The purpose would be to provide a hierarchial directory structure and to provide better support for large disks (with more than 65,000 blocks). It is expected that the method used to indicate a particular directory would be through the use of a PATH command that would allow a logical device name to be associated with a particular directory structure. RT-11 format devices would continue to be supported as well as the new TSX-Plus file structure.

The third extension under consideration is to adapt TSX-Plus for operation on 11/44 and 11/70 computers with 22-bit memory mapping. Preliminary investigations indicate this is feasible but may require some changes to device handlers.

S&H bulletin

VOLUME TWO, NUMBER FIVE APRIL 1984

INDAS VERSION 1.3 RELEASED

Result of 10 Man-years of Development and a Year's Field Testing

INDAS [Integrated Data Analysis System] Version 1.3 is now in distribution, after a year of field testing at more than 30 sites and an investment of 10 man-years of development effort. INDAS is a complete data analysis system for Digital Equipment Corporation VAX/VMS systems. Using the integrated INDAS programming language, database facilities, and powerful statistical procedures, the data analyst can perform the complete conversion of raw data to final printed results within the INDAS system.

The comprehensive set of procedures for statistical analysis includes descriptive and univariate statistical analysis; frequency tables; general linear model including ANOVA, MANOVA, and analysis of covariance; non-parametric analysis of variance; t-tests; probit analysis; multiple, stepwise, and non-linear regression; maximum likelihood estimation; spectral analysis; correlation (Pearson, Spearman, Kendall); canonical correlation; discriminant analysis; cluster analysis; and factor analysis. Graphics facilities include procedures for generating curve and contour plots, bar charts, pie charts, star charts, and block charts.

The INDAS database facility includes join, merge, concatenate, sort, and select operations. The database dictionary allows dynamic modification of variable names, labels, and formats.

Sophisticated users can use the INDAS Matrix Manipulation Language to develop new statistical procedures. The Matrix Manipulation Language is similar in power to APL (allowing full dynamic matrix operations), but does not require the use of a special terminal.

TSX-Plus FEATURED IN DECUS SCHEDULE— CINCINNATI '84

According to the preliminary DECUS schedule information we have received, TSX-Plus will be featured in 5 presentations of the June 4th through June 8th symposia.

The first TSX-Plus paper, "TSX-Plus Shared Run-time Systems", will be delivered Monday evening.

The Wednesday sessions will include two TSX-Plus presentations: "TSX-Plus Internals" and "TSX-Plus Magic". "TSX-Plus Internals" will be delivered by Jan Bramlet of S&H Computer Systems, Inc. This presentation will cover the internal structure and organization of TSX-Plus with special emphasis on memory usage, data structures, and system values associated with performance.

On Thursday, another two presentations will feature TSX-Plus. The first of these will be "RT-11/TSX-Plus Compatability Issues". The final TSX-Plus paper to be delivered at the Spring DECUS will be "TSX-Plus Real-time I/O Techniques".

For those who are interested in the DEC Professional computer, there will be a "PRO-350 Question and Answer Session" and a Digital tutorial dealing with the transportation of RT-11 programs from PDP-11's to the PRO-350 on Wednesday.

ENHANCEMENTS FROM THE USER WISH LIST UNDER CONSIDERATION

In response to user interest and requests, our development staff is investigating the feasibility of supporting the DEC DHV11 under TSX-Plus. Also under consideration is the development of a version of TSX-Plus to run on the DEC Professional-350.

S&H TO HOLD TECHNICAL/MARKETING SEMINAR IN LONDON

Following the DEC User Show at the Cunard International Hotel (15-17 May), Phil Sherrod will present a technical seminar on TSX-Plus. This seminar will concentrate on the architecture of TSX-Plus version 5.0 and will provide time for questions. Richard Dohrmann will also present a portion of the seminar devoted to S&H's marketing plans.

TSX-Plus V5.0 ADDITIONAL CORRECTIONS AND ENHANCEMENTS

The following additions and corrections were included in TSX-Plus version 5.0 after printing of the TSX-Plus manual and release notes.

A bogus flag setting was corrected which indicated the need for an outswap for a job that had requested memory expansion when no job could be found having issued the request. The most frequent symptom was that outswapped jobs would remain out of memory and only jobs currently residing in memory were able to execute.

A problem was fixed which caused all ACCESS specifications to be ignored if any invalid device specification was encountered. All valid ACCESS specifications are now retained.

The virtual image bit is now maintained outside the normal job region preventing a job which is loaded as a virtual job from triggering job mapping changes by alteration of the JSW bit.

The patch released for the rebuilding of the device driver, DD, requires an additional definition to the TSXCND file. This definition, TSX$P=1, is used to trigger the inclusion of TSX-Plus specific code.

An alteration has been made in device status bit checking during a USR request. The non-file directory structure random access bit is checked first, then the special directory structure bit is checked.

A problem was corrected in the job memory allocation for a system generated with job swapping disallowed (SWAPFL set to zero). This problem could cause system malfunction when two jobs were being started simultaneously and there was only memory available for one.

A problem was correced which prevented the IND special symbol ⟨EXSTAT⟩ from being correctly set after an error.

TSXLIB UPDATED

Like RT-11, TSX-Plus offers the MACRO programmer a number of system services via programmed requests or EMTs. RT-11 makes its system services available to the FORTRAN programmer through the system subroutine library, SYSLIB. TSXLIB makes the TSX-Plus EMTs available to the FORTRAN programmer as a library of callable routines. The package includes the MACRO source modules for all the routines, a user's manual in machine readable form, a cross reference chart, an indirect command file to build the library, and the implemented library.

The library has been updated to include all EMTs through TSX-Plus V5.0. It is available from the DECUS Program Library (order #11-490) on RX02 Floppy Diskette (KA) and 600 ft Magtape (MA), both in RT-11 format. The address for the DECUS Program Library is:

DECUS
One Iron Way
Marlboro, MA 01752

With this release, maintenance of TSXLIB has been assumed by NAB Software Services, Inc. of Albuquerque, NM.

TSX V10A NOW SPECIAL-ORDER

Version 10A of TSX, released in 1980, is now available by special-order only with an unsupported license. This product was the predecessor to TSX-Plus and is for use with LSI-11 and PDP-11 machines which do not support memory management. For prices and other information, contact S&H sales staff.

CTS-300 VERSION 8.0 PATCH FOR EXECUTION UNDER TSX-Plus

The following patch is necessary for executing CTS-300 version 8.0 under TSX-Plus. As distributed, DIBOL will attempt to open the error message file "SY:ERMSG.TXT" on an illegal TSX-Plus channel number. When the open fails, the error message "?INI-E3" results. The patch will correct the SUD.RTS file resident on SY:. DO NOT PATCH THE ORIGINAL DISTRIBUTION. BACKUPS SHOULD BE TAKEN BEFORE INSTALLATION OF THE PATCH.

```
R SIPP                  4216
SY:SUD.RTS/A            4016
0                        ^Z
4146                    14024
416                     4016
 ^Z                      ^Y
                         ^C
```

PATCH TO ALLOW COBOL-Plus VERSION 5.0 TO EXECUTE UNDER V4 RT11SJ

A minor problem has been discovered when using COBOL-Plus version 5.0 with the RT-11 version 4 single job monitor. In order to function as a virtual job under RT-11 version 5, the VIRT$ flag (mask 2000) in the SAV image Job Status Word has been set for the COBOL-Plus compiler (COBOL.SAV) and linker (CBLINK.SAV). This flag is ignored under the RT-11 version 5 SJ monitor, but is incorrectly interpreted by the version 4 SJ monitor as indicating an XM program. When the COBOL-Plus version 5.0 compiler is run under the RT-11 version 4 SJ monitor, the following error message (and a corresponding message for CBLINK.SAV) is generated:

?KMON-F-Extended memory monitor required for SY:COBOL.SAV

These programs can be successfully run with the RT-11 version 4 SJ monitor by clearing the VIRT$ flag in the program SAV image Job Status Word. The two following examples indicate appropriate patch to the COBOL-Plus version 5.0 compiler and linker to permit their use with the RT-11 version 4 SJ monitor. This patch does not affect their execution in any way when running under the RT-11 version 4 SJ monitor. Also, this patch is not necessary and should not be applied when using COBOL-Plus version 5.0 with RT-11 version 4 XM, any RT-11 version 5 monitor, or TSX-Plus.

```
.R SIPP
*SY:COBOL.SAV
Segment? 0
Base?    0
Offset?  44

Segment   Base    Offset   Old     New?
000000    000000  000044 003000   1000
000000    000000  000046 000000    ^Y
* ^C
```

```
.R SIPP
*SY:CBLINK.SAV
Base? 0
Offset? 44

Base      Offset   Old      New?
000000    000044   002000      0
000000    000046   000000    ^Y
* ^C
```

USE OF SHARED RUN-TIME SYSTEMS WITH COBOL-Plus VERSION 5.0

COBOL-Plus programs running under TSX-Plus may use either a shared run-time system or a non-shared version. Explicit selection for whether COBOL-Plus programs will use the shared or non-shared run-time system may be made by including switches when linking the COBOL-Plus program. The CCL switches are:

Switch	Result
/SHARED	Use the shared run-time library
/NOSHARED	Use the non-shared run-time

The switches to use when running CBLINK directly are:

Switch	Result
/L	Use the shared run-time library
/N	Use the non-shared run-time

The appropriate default (shared or non-shared) should be selected automatically when linking COBOL-Plus programs. If the COBOL-Plus shared run-time system is loaded, then CBLINK should link the COBOL-Plus program to use the shared run-time, if not, then CBLINK should link the program to use the non-shared run-time. However, COBOL-Plus version 5.0 does not correctly perform this automatic run-time selection. In order to use a shared run-time system with COBOL-Plus programs compiled and linked under COBOL-Plus version 5.0, it is necessary to explicitly select the appropriate switch. If you wish automatic selection of the shared run-time, then it is necessary to apply a patch to CBLINK.SAV version 5.0.

In order to automatically select use of a shared run-time for COBOL-Plus programs linked under COBOL-Plus version 5.0 whenever CBLINK locates the COBOL-Plus shared run-time during its execution, execute the following command file:

RT-11 V4	RT-11 V5
R SIPP	R SIPP
SY:CBLINK.SAV	SY:CBLINK.SAV/C
0	0
11356	11356
140426	140426
^Y	^Y
^C	136540
	^C

SPOOLER: INTERACTION OF HOLD COMMANDS WITH PENDING FILES

It has been brought to our attention recently that files can get hung up in a print spooler queue even when the printer is on line and idle and the correct form is mounted. This situation arises when there are files pending in the spooler queue and someone issues a SPOOL dev,HOLD command. The result is that files queued after the current file will not print upon completion of the current file. Further, files subsequently sent to the spooled device are printed normally, although ahead of those already in the queue. The solution is to issue a SPOOL dev,NOHOLD command. Pending files will then be printed. When the spool queue is empty, then the SPOOL dev,HOLD command may be reissued.

This situation commonly arises when log-on command files contain a SPOOL dev,HOLD command. Then, whenever one of these users logs on, any pending files in the spooler queue will remain in the queue until the SPOOL dev,NOHOLD command is issued. The solution is: DO NOT ISSUE UNNECCESSARY SPOOLER COMMANDS. That is, do not include spooler commands in start-up command files. The default spooler hold/nohold status is selected with the SPOOL macro during TSX-Plus system generation. As spooler commands are effective system-wide, we recommend that only someone near the printer issue spooler commands. That operator should also consider the effect of spooler commands on current, pending and future print requests.

bulletin

VOLUME TWO, NUMBER SIX JUNE 1984

TSX-Plus V5.1 SET FOR SEPTEMBER RELEASE

TSX-Plus for PRO-350 in Field Test

Currently under field test is a version of TSX-Plus which supports the DEC Professional-350 personal computer. When running on the Pro, TSX-Plus may use the console, the printer port, and the communications port as time-sharing lines, thus providing support for up to 3 time-sharing users. The printer port and the communications port may also be used as I/O lines. The CL facility (see below) should be used to drive the serial printer and the communications port with VTCOM rather than the LS and XC handlers.

The PI.TSX handler is used to control the console terminal on the Pro. The handler must be on the system disk when TSX-Plus is started. It is not loaded as a handler but rather as a shared run-time system.

The Pro version of TSX-Plus must be started under RT-11/FB (not XM). Because of the speed of the disk on the Pro, it takes about 15 seconds for TSX-Plus to get started after the "R TSX" command is issued.

If the printer port is used as a time-sharing line, a cable must be constructed to connect it to a terminal. The following pins on the printer port are significant: 2 (transmit data), 3 (receive data), 6 (data set ready), 7 (signal ground). In addition, pins 8 and 9 must be asserted by the terminal.

The communications port has a standard DB25 (RS232) connection. Modem support is provided for the communications port.

Release of the TSX-Plus Pro version will coincide with the release of TSX-Plus 5.1 and contain all of the functionality of that version.

NEW COMMUNICATIONS LINE (CL) HANDLER IN TSX-Plus 5.1 Will Allow Time-sharing and Communications I/O on Same Multiplexer

A Communications Line (CL) handler is included with version 5.1. The CL handler allows Input/Output operations to be performed to serial communication lines connected to DL11, DLV11, DZ11, and DZV11 communication controllers. With the CL handler it is possible to have some lines on a multiplexer used as TSX-Plus time-sharing lines, and other lines on the same multiplexer used to drive I/O devices such as printers, plotters, and modems. An individual line is declared to be a time-sharing line or a communications I/O line at sysgen time. Up to 8 communication lines can be controlled by the CL handler. Some of the important features of the CL handler are summarized below:

Up to 8 communication lines may be controlled through the CL handler. The device names are "CL0:", "CL1:",... "CL7:". The lines may be connected to any combination of DL11 and DZ11 communication controllers and may share the same multiplexer controllers as TSX-Plus time-sharing lines.

Internal queueing is used within the handler to allow concurrent Input/Output operations to be performed on all of the lines.

The CL handler allows both input (read) and output (write) operations. Full duplex (simultaneous) read and write operations may take place on each line.

The communication lines may be used with the TSX-Plus spooling system to allow spooled output to devices on communication lines.

The CL handler responds to XON/XOFF (ctrl-Q/ctrl-S) control characters to stop and start its transmission and will generate XON/XOFF characters to control the speed of a device transmitting to a CL line.

A "binary mode" is available for CL lines to allow full 8-bit, transparent I/O to devices.

Modem control is supported. Ring and carrier detect signals may be monitored and data terminal ready (DTR) can be controlled by a program or SET command.

The CL handler is implemented as a system virtual overlay, minimizing the amount of code and data that is required in the unmapped portion of the system. Typically the size of the unmapped portion of the CL handler is 200 to 500 bytes (depending on the number of lines supported) and the mapped portion is approximately 2200 bytes.

The CL handler can be used as a replacement for the LS and XL handlers (the XL handler is used with the RT-11 VTCOM program).

Two types of CL lines are available. If a CL line is defined during system generation (in much the same method as a time-sharing line definition), the communication line unit is dedicated and cannot be used as a time sharing line. If the CL unit is reserved during system generation but not defined as a physical line, a SET command can be used to attach a normal

time-sharing line to the communication line unit. A CL unit may not be assigned to a time-sharing line which is active (a user logged on) or which has another CL unit assigned.

NEW TSX-Plus PROGRAM DEBUGGING FACILITY Included With TSX-Plus Version 5.1.

This new facility replaces the old TSODT program. The major features of the new debugging facility are listed below:

The new debugger does not share memory space with the program being debugged (it is a system overlay). This means that the debugger can be used with programs using up to 64Kb of memory. It can also be used with real-time programs that access the I/O page and with programs that use shared run-time systems or PLAS regions.

The debugging facility is invoked by use of the "RUN/DEBUG" keyboard command. It is not necessary to link the debugger with the program being debugged.

The execution of a program can be interrupted at any time by typing control-D. This causes an entry to the debugging system (similar to hitting a breakpoint) where the status of the program can be checked and then execution continued if desired.

When enabled by a keyboard command (SET CTRLD DEBUG), the debugger may be invoked by typing control-D even during execution of a program which was not started under control of the debugger.

A data "watchpoint" facility is included which allows a data item to be monitored and a break to occur when the value of the data item changes.

Program traps to 4 and 10 are caught by the debugger rather than causing a program abort and entry to the keyboard monitor.

The commands and syntax of the debugging facility are similar to those used by ODT. However, not all ODT commands are implemented and some additional features are provided.

A program is started with the debugger by typing

RUN/DEBUG program or R/DEBUG program

When the program is started, the following prompt is printed at the terminal:

TSX-Plus debugger

DBG:

The "DBG:" prompt indicates the debugger is in control and is waiting for a command. On entry to the debugger, the program start address is in R0 and is set so that the ";G" command can be used to initiate execution without having to specify a starting address. At any time during the execution of a program with the debugger, the execution of the program may be interrupted by typing control-D. This causes an entry to the debugger similar to a breakpoint. The status of the program can be checked and its execution resumed by typing ";P". If a system service call (EMT) is being executed when control-D is typed, the service call runs to completion and the debugger is entered as the system is about to return from the EMT to the program. Normally, control-D only causes a breakpoint if the executing program was started with the debugger. However, it is possible to enable control-D to cause a debugger breakpoint for any program, even if the program was not started with the debugger. To enable control-D to interrupt the execution of any program run from your time-sharing line, issue the following keyboard command:

SET CTRLD DEBUG

To disable this function, use the following command:

SET CTRLD NODEBUG

If the debugging facility is wanted, the DBGFLG sysgen parameter in TSGEN must be set to 1 (one) when the system is generated. The debugging facility adds approximately 3.3Kb of code to the mapped portion of the system if it is included.

SPOOL COMMAND IMPROVED IN TSX-Plus V5.1 Allows Individual Deletion of Spool Files

The spooling system has been enhanced to allow deletion of spool files that are pending in the spool queue. The new form of the SPOOL delete command is:

SPOOL dev,DELETE [id]

Where dev is the name of a spooled device and id is an optional file identification number used to specify which file in the spool queue is to be deleted. The SHOW QUEUE command has been changed to list file identification (ID) numbers along with other spool file information.

When an identification number is specified with the SPOOL DELETE command, that number identifies the file to be deleted. In this case, the spooled device (dev) may be the name of any spooled device and does not have to match the device for which the file is pending.

If the spool file identification number is omitted, the SPOOL DELETE command deletes the file which is currently being printed on the specified device.

If the specified file is currently open, the file is marked to be deleted but is not actually deleted until it is closed and the spooler begins to process it. If the file is currently being printed when the delete command is issued, the spooler deletes blocks as they become available and the spooled device is tied up until the file deletion is completed.

TSX-Plus V5.1 .FORK REQUEST PROCESSING ENHANCED

An improvement has been made in the handling of .FORK requests for interrupts. The improvement reduces the latency time between the occurrence of an interrupt and the time that

a real-time interrupt service routine is entered. This same improvement also reduces the likelihood of having a "No free fork blocks" system error while doing high speed input to a terminal line.

IB DEVICE HANDLER SUPPORTED IN V5.1

Support is now included for the IB device handler which controls the DEC IB(V)11 interface card for IEEE GPIB devices. This required the addition of an executive routine to TSX-Plus which relocates a user's virtual address to a PAR6 mapping value and boundary. To interface the IB routines to TSX-Plus, changes are required to both TSGEN and the IB handler. The change in TSGEN is necessary since the normal IB subroutines attempt to open the IB device on decimal channel numbers 16, 17, 18, and 19. Increasing the TSGEN parameter "NUCHN" from 16 to 20 (decimal) allocates additional channels and allows the IB subroutines to execute without changes. (It may be necessary to allocate more channels in non-standard configurations.) The change in the IB handler is due to the use of PAR6 for mapping to the user buffer by TSX-Plus rather than PAR1 as in RT-11. This requires the following three changes to the IB handler:

	old value	new value
PAR1 =	172342	172354
PAR1HI =	37776	157776
PAR1LO =	20000	140000

TSX-Plus V5.1 PROBLEM CORRECTIONS

The following corrections are included in TSX-Plus version 5.1:

A problem has been corrected which occasionally caused TSX-Plus to lose completion routine requests. This problem only occurred when more than one completion routine was pending for the same job. The most common symptom was that the VTCOM program would sporadically die while running under TSX-Plus.

A problem was corrected which caused echoing of carriage return and line feeds when echoing had been disabled with the "F" program controlled terminal option. This occurred when the single line editor was set both "ON" and "TTYIN".

A problem was corrected which caused the error message "MON-F-Invalid address as EMT argument" for valid I/O requests. The message resulted from checking the buffer address for a word (even address) boundary on all transfer requests. The corrected implementation is boundary checking only for devices which require word alignment such as cached devices, VM, and others.

A problem was corrected which occasionally resulted in corruption of the spooling system when message channels were included. An incorrect amount of memory had been reserved for message buffers.

A problem was corrected in VM which corrupted a memory location and caused unusual system errors when writing partial block transfers.

A problem was corrected so that the appropriate error level is passed to IND when invalid dates or times are entered.

The TSAUTH program has been corrected so that it is no longer possible to create duplicate PPN entries.

MEMORY USAGE by COBOL-Plus XM PROGRAMS

Starting with COBOL-Plus, version 5.0, it is possible to link programs so that all program and data segments will be memory resident by using the PLAS features of TSX-Plus version 4.1 or later or the RT-11 XM monitor. Certain restrictions apply to code and data segments of COBOL-Plus XM programs which, if not observed, can prevent these programs from starting. The resulting error message is:

?CBRTS-F-IXM
Program segments are too large to run as virtual job

The reason for this message is related to the way in which COBOL-Plus uses memory mapping for XM programs. Virtual job memory is organized into 8Kb pages in the PDP-11 architecture, of which only 8 pages can be simultaneously addressed. When a COBOL-Plus program is started, at least 4 pages (8Kb each) are reserved for use by the program root and the COBOL-Plus run-time system leaving 4 pages for program and data segments. (Programs using either shared or non-shared run-time systems have the same XM mapping restrictions.) Part of the 4 root pages contains a table of program and data segments used by the automatic segmentation feature to control the page mapping in an XM program. In a program with a large number of segments, this table may extend above the 4 root pages, reducing the number of pages for program and data segments to 3 (or possibly fewer in gigantic programs).

Of the 4 (or fewer) pages left for program and data segments, at least one will be used for data segment mapping and at least one will be used for program segment mapping. The amount of memory used for segment mapping is determined by the largest segment of each type. If the largest data segment is 5Kb, then 1 page will be used for data segment mapping. If the largest data segment is 9Kb, then 2 pages will be used for data segment mapping. Similar calculations hold for program segments.

Segment mapping conflicts may arise in large programs with a large number of segments and/or large code or data segments. XM programs cannot be started if the number of pages needed for mapping the root+the largest data segment+the largest code segment exceeds 8. For example, if a program had a largest data segment of 17Kb, and a largest code segment of 9Kb, it could not be started. Mapping for the data segments would require 3 pages; mapping for the code segments would require 2 pages; and mapping for the root always requires at least 4 pages. Since the total of 9 pages needed exceeds the limit of 8, the program would fail with the run-time error described above. The situation can be even more complex if the total number of pages needed for mapping code + data segments only totals 4 and there are numerous segments. Because of the size of the segmentation table, some XM programs with a large number of segments may not start while another program with larger code and data segments will.

Because data divisions may only be segmented at the 01 level, large tables or arrays are never separated into multiple

segments: Segmentation of code sections may be controlled to some extent with the compiler "/X:value" switch. Segment size information can be obtained with the compiler "/I" switch.

When attempting to load large COBOL-Plus programs into XM regions, it is important to achieve a compromise between the total number of segments and the size of the largest code and data segments.

THE 11/73 UPGRADE BLUES

There are now a fair number of TSX-Plus sites using the new 11/73 processor and we have accumulated enough experience to make some general comments about upgrading. First, the 11/73 is indeed faster than the 11/23 processor and it is usable with TSX-Plus. Depending on the specific application, the 11/73 seems to have an overall speed advantage about three times over the 11/23. Of course, applications which are I/O intensive will see much less dependence on the processor speed. When programs are bound by computation speed, upgrading the processor is very effective. When they are bound by disk access, then using the general data caching facility available in TSX-Plus version 5.0 is more effective. Sites which are using general data caching and have upgraded from the 11/23 to the 11/73 processor have generally been favorably impressed with the performance improvement. However, the transition to the 11/73 has not been without difficulty at some sites.

The 11/73 processor draws twice the power as that of the 11/23. If the total power consumption exceeds the capacity of the power supply, then random errors can occur. When there are DC power fluctuations, the program counter seems to be the first casualty. This may result in the system halting at location 000002 or cause a Kernel Mode Trap with a very small number (less than 001000) as the argument value. The general symptom of a marginal power supply is random errors, such as Kernel Mode Traps with variable argument values, and Job Number Zero and Stack Overflow errors.

Some of you have non-DEC disks and controllers. Because of the 11/73's faster clock speed, there is potential for bus timing errors with older DMA device controller designs not intended for use with faster clocks. In discussions with several disk controller manufacturers, we have discovered that some of these devices are not "qualified" for use with the 11/73 processor. Not "qualified" seems to mean in hardware jargon that the manufacturers haven't tried it and don't really know whether it will function correctly or not. Part of the problem may be that the 11/73 specifications have not yet been widely disseminated.

We know of at least two non-DEC disk controllers which did not initially function correctly with the 11/73 board. In one case, we were able to overcome a specific problem with a minor patch to the device handler. In another case, because of the different line-time-clock built into the 11/73 board, it was necessary to disable the clock line on a controller card. It is not obvious to us that such relatively simple modifications will enable all DMA devices to function with the 11/73. Our recommendation is to beware of compatibility claims.

One other interesting feature of the 11/73 which has surprised some is that it contains an odd-address trap while the 11/23 did not. That is, when a word-oriented instruction is supplied an odd address, the 11/23 would truncate the address and continue, whereas the 11/73 will generate an odd address error. This has resulted in the identification of programming errors in some long-standing code. If the error is in a user program, an odd-address trap may manifest as: "?MON-F-Trap to 4". If the error is in a special device handler, it will probably result in a Kernel Mode Trap.

When upgrading an 11/23 to an 11/73, it is also common to upgrade the amount of memory to more than 256Kb. When doing so, keep in mind that not all devices or device handlers support 22-bit addressing. The only DEC standard devices and handlers that do support 22-bit addressing are: DL with the RLV12 controller (and both 22-bit jumpers strapped); DU and all the devices it supports; and MS using the TSV05. Some non-DEC devices also support 22-bit addressing with the appropriate device handler. While 22-bit errors may be manifested in several ways, the most common is the message "Device I/O error". If TSX-Plus works correctly when the TSGEN parameter EXTMCH=0, but fails in some way when EXTMCH=1, then you almost certainly have either a device handler that does not support 22-bit addressing, or a device controller that does not support or is not properly configured for 22-bit addressing. If you want to use more than 256 Kb of memory in cases where 22-bit support is not available, then use the MAPIO parameter to the DEVDEF macro for that device in TSGEN.

We are still learning about the 11/73. If you have a solution to an upgrade problem that might help us ease the transition for someone else, let us know, preferably by mail.

JPY ASSOCIATES LIMITED

9.83SRM.DOC

SUPPORT SERVICES
from
JPY ASSOCIATES LIMITED

Supported Products:

The Customer gets telephone support (during normal working hours) as well as written responses to written problems.

To clarify: whenever possible an answer is given by return. However not every problem can be solved in one short conversation. So it may be necessary for us to work on the solution using our own system. If this is the case then we would normally expect the user to write the problem down and send it to us. A quick analysis of the problem on the initial telephone call can be made so that the Customer is clear what information is required of him.

If the problem proves to be lengthy, the Customer may wish to opt for on-site assistance and our consultancy services would then apply.

The Customer is also entitled to updates when they are available (at media, shipping and handling cost only).

Terms of Support:

JPY Associates agrees to provide the Customer with support as defined above on behalf of the Supplier.

CUSTOMER NAME: Technische Hogeschool Twente Site Address: Centrale Ontvangst Hal D Hallenweg The Netherland

Product	Licence Number	*Current Version	Support Period From:	To:	Cost
TSX+	1606	5.0	14.8.1984	13.8.1985	£506.00

Signed for and on behalf of
JPY Associates Limited

Date:

N.B. This statement will be signed upon receipt of support renewal fee from the Customer. Please retain the signed statement as proof of your support entitlement.

* Current version of the product existing, not necessarily the version run by Customer.

138 High Street, New Malden, Surrey, England, KT3 4EP
Telephone: 01-949 1088 Telex: 8954665 (Ref JPY)
Attn Keith Pegler
Directors: J.P. Yardley, B.Sc., Ph.D., C.Eng., M.I.E.E., M.B.C.S., A.E. Wheatley, B.Sc., Ph.D.
Company Secretary: P.J. Miller
Registered No.: 1601775 England, V.A.T. No. 317 9784 16, Registered Office: 76 Cambridge Road, Kingston upon Thames, Surrey

s&h computer systems, inc.

ACKNOWLEDGMENT OF MAINTENANCE AND/OR SUPPORT

CUSTOMER

Name Technische Hogeschool

Address Twente

Country/State

ZIP/TELEX TEL

Contact

DISTRIBUTOR

Name JPY Associates

Address 138 High Street

New Malden Surrey

Country/State England

ZIP/TELEX 851-8954665

Contact

S&H Computer Systems, Inc. hereby acknowledges receipt of a fee from Customer for which Distributor (or if purchased directly from S&H, S&H) agrees to provide Customer with the S&H maintenance and/or support product(s) specified below as defined in the S&H standard price list. This maintenance and/or support is effective the date of this receipt and according to the terms of the S&H Software License Agreement between Customer and S&H.

MAINTENANCE AND/OR SUPPORT PRODUCT[S]

CODE	DESCRIPTION	LICENSE NUMBER
XP-P-DX	TSX-PLUS support	56-SP-1606
	Support period from	
	7-31-84 to 7-31-85	

Authorized S&H Representative

Victoria J. Chambers

TSX-Plus Version 5.0 Release Notes Addendum

The following additions or corrections were discovered following the printing of the manual and release notes.

1. A problem was corrected concerning a bogus flag setting indicating the need for an outswap for a job requesting memory expansion when no jobs could be found which issued the request. The most likely symptom was that outswapped jobs would remain out of memory and only jobs which currently resided in memory could execute.

2. A problem was fixed which ignored all ACCESS specifications if any invalid device specification was encountered. The correction retains all valid ACCESS specifications.

3. The virtual image bit is now maintained outside the normal job region so that a job which is loaded as a virtual job will not trigger job mapping changes by altering the JSW bit.

4. If it is necessary to rebuild the DD device driver, the patch released requires a definition which was omitted from the TSXCND settings. The following additional definition must be made in TSXCND:

```
TSX$P   =       1
```

   This conditional is used to trigger the inclusion of TSX-Plus specific code.

5. An alteration was made dealing with the device status bit checking during a USR request. The non-file directory structure random access bit is checked first and the special directory structure bit is then checked.

6. A problem was discovered and corrected in the job memory allocation for a system generated with no job swapping allowed (SWAPFL set to zero). This error could cause system malfunction when two jobs were being started simultaneously but only enough memory was available for one.

7. A problem was corrected which prevented the IND special symbol <EXSTAT> from being correctly set after an error.

TSX-Plus Version 5.0

Release Notes

These release notes describe the differences between TSX-Plus version 5.0 and version 4.1. The TSX-Plus Reference Manual and TSX-Plus System Manager's Guide incorporate the information described in earlier release notes. In case of any differences between the release notes and the manuals, the release notes take precedence.

February, 1984

CONTENTS

1. The output portion of the terminal driver has been rewritten and is approximately twice as fast as the old version. However, the following features were deleted from the new version:

   a) The SET TT LENGTH command is no longer functional. Output to the terminal will no longer be automatically suspended when the end of page is reached. The control-S and control-Q keys must be used to suspend and restart terminal output. (The NO SCROLL key on VT100 terminals alternately transmits control-S and control-Q.)

   b) The SET TT FILLER command is no longer functional. It is no longer possible to specify filler (delay) characters to be inserted after carriage return or line feed.

   c) VT52 "hold screen" mode is no longer supported and the SET TT [NO]HOLD command has been removed.

   d) The ACK/ETX control sequence for old Diablo terminals is no longer supported. Note that newer Diablo terminals which use control-S/control-Q to synchronize transmission still work correctly.

2. Keyboard commands may now be easily defined by the time-sharing user. New keyboard commands may be defined in a manner similar to that used by VAX VMS:

```
name :== string
```

   where "name" is a 1 to 10 character command keyword which may consist of letters, digits, and the underscore symbol ("_"), and "string" is a string of up to 80 characters which defines the body of the command.

   For example, the following command would define the keyword "NOW" as being equivalent to the SHOW TIME command:

```
NOW :== SHOW TIME
```

   Multiple commands may be included in the body of a command definition by separating them with the backslash character ("\"). For example:

```
NOW :== SHOW DATE\SHOW TIME
```

   An up-arrow character ("^") may be included in the command body to cause all text on the command line following the command keyword to be inserted in the command body at the position of the up-arrow. For example, if a command is defined as follows:

```
NAMES :== DIR/ORDER:NAME ^
```

Then, if this command is invoked by typing:

```
NAMES *.MAC
```

The resulting command will be:

```
DIR/ORDER:NAME *.MAC
```

More than one up-arrow character may occur in the command body. The parameter string specified when the command is invoked is substituted for each occurrence of the up-arrow character.

A user defined command may invoke other user-defined commands within its definition provided that the definition does not call on itself and provided that the other commands are defined at the time the command is issued. For example, the following command definitions would be legal:

```
NOW :== SHOW DATE\SHOW TIME
STATUS :== NOW\SYSTAT
```

But, invoking either of the following pair of commands would cause an infinite loop:

```
ONE :== TWO
TWO :== ONE
```

It is possible to allow abbreviation of user-defined commands. To do this, place an asterisk character ("*") within the keyword at the point of the minimum length abbreviation. For example, the definition:

```
ST*ATUS :== NOW\SYSTAT
```

Would allow the STATUS command to be abbreviated to "STAT" or "ST" but not to "S".

You can specify the order in which the TSX-Plus command interpreter checks for user-defined commands by use of the SET UCL command. This command has four forms:

```
SET UCL FIRST
SET UCL MIDDLE
SET UCL LAST
SET UCL NONE
```

If the SET UCL FIRST command is used, user-defined commands will be processed before system commands. This allows user-defined commands to replace system commands but makes the processing of system commands slower.

If the SET UCL MIDDLE command is used, user-defined commands are processed after system commands but before checking for command files

and SAV files with names that match the command keyword. Using this setting, it is not possible to replace a system command with a user command but both system commands and user-defined commands are processed quickly.

If the SET UCL LAST command is used, a command will not be checked to see if it is a user-defined command until after it is checked to see if it is a system command, the name of a command file on DK, the name of a command file on SY, or the name of a SAV file on SY. Using this setting, it is not possible to replace a system command with a user command and user commands cannot have the same name as command files or SAV files. System commands are processed quickly (the same speed as SET UCL MIDDLE), but the processing of user-defined commands is slow.

If the SET UCL NONE command is used, user-defined commands are never interpreted. In this mode, attempts to invoke user-defined commands will result in the error:

```
?KMON-F-Unrecognizable command
```

The following list illustrates where the FIRST/MIDDLE/LAST setting causes the command interpreter to check for and process user-defined commands:

```
FIRST  -->
            See if command is a system command
MIDDLE -->
            Look for command file on DK: with command name
            Look for command file on SY: with command name
            Look for SAV file on SY: with command name
LAST   -->
```

If an underscore character ("_") is typed in front of a command name, this is a signal to the system that the command is not to be interpreted as a user-defined command. For example, if the NOW command has been defined as shown above, then the following command will be recognized as a user-defined command and will cause the date and time to be displayed:

```
NOW
```

However, the following command would not be recognized as a user-defined command and would be rejected as an undefined command since there is no NOW command defined by the system:

```
_NOW
```

There are two reasons for using the underscore prefix. If UCL has been set FIRST, the underscore prefix can be used to speed up the processing of large numbers of system commands that could be present in frequently executed command files. For example, the NOW command could be defined as follows to cause it to execute faster:

```
NOW :== _SHOW DATE\_SHOW TIME
```

The second and more important reason for using the underscore prefix is to allow user-defined commands to be defined which replace system commands but use the system commands in their definitions. (Note that it is necessary to SET UCL FIRST to allow system command replacement.) For example, the following definition replaces the system DIRECTORY command with a user-defined command that has the same name but which always orders the files alphabetically:

```
DIR*ECTORY :== _DIR/ORDER:NAME ^
```

If the body of this command were not preceded with the underscore character, then it would be a circular definition and cause an infinite loop.

A user-defined command can cause a command file to be executed. However the command file invocation must be the last (or only) command defined within the body of the command. For example, the following definition causes all BAK files to be deleted, and a command file named LOGOFF to be executed when the OFF command is used (the LOGOFF command file could end with an "_OFF" command to do the actual logoff):

```
OFF :== _DEL *.BAK/NOQ\@LOGOFF
```

A user-defined command may be replaced at any time by simply entering a new definition for the command. A command may be deleted by entering its name and ":==" without a command body. For example, the following command deletes the definition of the NOW command:

```
NOW :==
```

The SHOW COMMANDS keyboard command may be used to display a list of all current user-defined commands.

Commands defined by one time-sharing user are "local" to that user and do not affect other users. However, when a virtual line is started the virtual line "inherits" the user-defined commands that are in effect for the primary line (relative line number 0; ^W0) at the time that the virtual line is started. User-defined commands created while on a virtual line are not available from other virtual lines or from the primary line. All user-defined commands for a job are reset (forgotten) when the job logs off.

The processing of user-defined commands is performed by a program named TSXUCL.SAV which is now supplied with TSX-Plus. This program must be copied to the system disk if user-defined commands are to be used.

There are three system generation parameters related to user-defined commands. The first is named U$CL. This parameter must be set to 1 to enable TSXUCL to be called.

The second parameter is UCLMNC which specifies the maximum number of user-defined commands that can be defined by each job. The size (in blocks) of the file used to hold the user-defined commands (SY:TSXUCL.TSX) is approximately equal to the value of this parameter times the total number of jobs divided by 5.

The third parameter, UCLORD, specifies the default order in which user defined commands are to be processed. Equating this parameter to one of the symbols FIRST/MIDDLE/LAST/NONE selects the system default order for command interpretation. The SET UCL command may be used to change the order of command interpretation by each job.

3. A User Command Interface (UCI) facility has been added. The UCI facility allows a user-provided program to take over command acquisition from the TSX-Plus keyboard monitor. When the UCI facility is engaged, the user-written program will be called by the TSX-Plus keyboard monitor each time it needs a new command. It is the responsibility of the user program to prompt for the command and accept it from the terminal. This facility could be used to create a menu-style user interface to the system.

The following keyboard command is used to turn on user command interface processing:

```
SET KMON UCI[=file]
```

Where "file" is an optional file specification for the user-provided command interface program. If the equal sign and the file name are omitted, the default file spec is "SY:UKMON.SAV".

After this command has been issued, the TSX-Plus keyboard monitor will run the user-provided program each time it needs a new command. The program must prompt the user for a new command, accept the command, and perform the appropriate action. The program may pass commands to the TSX-Plus keyboard monitor by setting bit 5 (mask 40) in the Job Status Word and doing a .EXIT EMT (this does a "Special Chain Exit"). A command file may be passed to the keyboard monitor by returning a command of the form "@name". In this case all of the commands in the command file are executed by the keyboard monitor before it returns to the user command interface program for another command.

The UCI facility can be turned off by use of the following command:

```
SET KMON SYSTEM
```

4. A "Single Line Editor" facility is now available. This facility is generally compatible with the RT-11 "SL" editor and allows editing of the current input line using a subset of the KED functions and recall of the previous two input lines or fields. The TSX-Plus single line editor differs from the RT-11 single line editor in the following ways:

   a) The "Help" key (PF2) is not implemented.

   b) The following SET SL options are not implemented: ASK, LEARN, SYSGEN.

   c) The maximum length of a line or field that may be input with the single line editor is fixed at 80 characters. The SET SL WIDTH=n command is ignored.

   d) The line feed key deletes the word to the left of the cursor (like KED). Technically, characters to the left of the cursor are deleted until one of the following delimiters is reached: space, tab, comma, or equal sign.

   e) The up-arrow key can be used to retrieve either of the last TWO lines. Pressing the up-arrow key cycles between recalling the last line and the line prior to that.

   In addition to the RT-11 compatible features, the TSX-Plus Single Line Editor also provides a "KED compatible" mode of operation which places the numeric keypad in "function key" mode and provides additional editing functions. To enable the KED compatible mode, use the following SET command:

```
SET SL KED
```

   To disable the KED mode and return the numeric keypad back to numeric mode use the following command:

```
SET SL NOKED
```

When in KED mode, the following keys on the numeric keypad have the functions shown (functions selected by pressing PF1 before the key are shown following "/"):

VT100	VT52	Function
0	0	Move to left end of line / Delete to right end of line
1	1	Advance or backup one word / Change case of character
2	2	Move to left or right end of line / Delete to right end
3		Advance or backup one character
4	4	Set forward direction / Move to right end of line
5	5	Set backward direction / Move to left end of line
	6	Delete character under cursor / Replace deleted char.
	9	Delete word under cursor / Replace deleted word
-		Delete word under cursor / Replace deleted word
,		Delete character under cursor / Replace deleted char.
ENTER	ENTER	Equivalent to carriage return

The single line editor may be used with either VT100 or VT52 terminals. If you wish to use the single line editor, set the TSGEN parameter SLEDIT to 1 and issue a SET SL ON command after logging on to TSX-Plus.

When the single line editor is being used by a job, the job is activated as each character is typed. This requires that the job be in memory for the single line editor to process each character and will place an additional burden on systems with restricted memory space. If the single line editor is not used, jobs are only activated when a field activation condition occurs.

The single line editor is implemented as a system overlay rather than as a device handler as used by RT-11. If the single line editor is included at system generation time it adds about 3Kb to the mapped portion of TSX-Plus.

Unlike RT-11, the TSX-Plus single line editor does not force input to a new line when SET SL TTYIN has been issued and input is being accepted using the .TTYIN EMT. Thus, the single line editor may also be used to edit data entry fields as well as system command lines.

The following conditions disable the single line editor:

a) SET SL OFF command.

b) Setting bit 4 (EDIT$; mask 20) in the Job Status Word.

c) Setting bit 12 (TTSPC$; mask 10000) in the Job Status Word.

d) Use of high-efficiency terminal mode.

e) Declaring VT100/VT52 escape sequences as TSX-Plus activation conditions.

f) Accepting input with the .TTYIN EMT (unless a SET SL TTYIN command has been issued).

Note that in order to use the single line editor with COBOL-Plus programs it is necessary first to SET SL TTYIN and then from within the program to CALL "ESCAPE-OFF".

5. A generalized data caching facility is included with version 5.0. Data caching is a technique for improving system performance by keeping in memory a "cache" of the most recently accessed blocks of data. Each time a read operation is performed a check is made to see if the requested data block(s) are in the cache. If so, the data is copied from the cache buffer to the receiving program buffer and no actual device input/output is done. Write operations update the data in the cache as well as writing to the I/O device.

Previous versions of TSX-Plus have had a specialized data caching facility which cached only blocks from files declared to be "shared" to the system. This facility is effective for speeding up application data-base files such as COBOL-Plus and DBL indexed (ISAM) files. However, since this facility only caches blocks from files that are declared to be "shared" it does not affect the performance of other programs.

The new generalized data caching facility caches blocks from all files that are on devices that are mounted by use of the MOUNT keyboard command. Thus the MOUNT command now performs three functions: (1) enables data caching; (2) enables directory caching; and (3) mounts logical subset disks.

To enable data caching, assign a non-zero value to the parameter named "CACHE" in TSGEN. This causes the data caching code to be included in the generated system and controls the number of blocks of memory allocated for data caching buffers. If data caching is not wanted, set the CACHE parameter to 0 (zero). If data caching is enabled, approximately 1800 bytes of code are added to the unmapped portion of the TSX-Plus system and approximately 528*CACHE bytes of data area are reserved in the mapped portion of the system.

A SET command is available to dynamically alter the number of blocks of data held in the data cache. The form of this command is:

SET CACHE value

This command does not alter the amount of space allocated for the data cache (that is directly controlled by the CACHE sysgen parameter), but can be used to cause the system to use less than the full cache area. Operator command privilege is required to use the SET CACHE command. The primary use of this command is to allow the system manager to

experiment with different cache sizes to determine the effect on system performance. Once an optimum cache size has been determined the CACHE sysgen parameter can be set to this value and the system regenerated. A "SHOW CACHE" keyboard command can be used to display the current number of blocks being held in the data cache.

The effectiveness of the data caching facility increases with the number of blocks allocated for the data cache. In systems with large amounts of memory is it reasonable to allocate several hundred blocks to the data cache. However it is not wise to allocate so much memory space to the data cache that job swapping is significantly increased due to limited memory space for time-sharing users.

The amount of improvement due to data caching also depends on the ratio of the processor (CPU) speed to the speed of the I/O device being cached. The effects of data caching are most pronounced when a fast processor is running with a slow I/O device. Data caching is not recommended for systems which are primarily bound by CPU utilization rather than I/O throughput.

Data caching can have a dramatic effect on the execution of overlayed programs if the cache is large enough to hold the overlay segments. FORTRAN and COBOL-Plus compilation times are typically reduced by 20% to 40% by data caching.

The following table shows typical cache "hit" rates as a function of the cache size (in blocks) for various language processors performing assemblies or compilations:

Cache size versus percent of blocks read from cache while performing assemblies and compilations

Cache Size	MACRO	FORTRAN	F77	COBOL-Plus	DBL	Pascal-2
20	2%	0%	23%	11%	5%	0%
35	3	1	23	21	9	0
50	4	1	23	82	10	5
75	14	2	24	83	25	8
100	36	2	24	84	45	9
150	48	4	27	84	55	90
175	49	51	33	87	84	90
200	50	87	33	87	84	90
250	66	90	34	87	84	90
275	92	92	35	88	84	91
300	92	93	87	88	84	92
400	92	94	94	95	84	92
500	92	97	94	98	84	93

The single job (non-XM) versions of F77 and Pascal-2 were used in making these measurements.

The following statistics for cache hit rates were measured while running a COBOL-Plus program performing 5000 random reads on an indexed organization (ISAM) file containing 44000 records with a 16 byte key.

Cache Size	Cache Hit Rates for Random Reads
5	24 %
10	32
15	38
20	46
25	50
30	55
40	60
50	64
60	65
70	67
80	70
90	71
100	72
200	79
300	82
400	83
500	84
1000	85

These statistics were gathered by generating a TSX-Plus system with a 1000 block data cache and then using SYSMON to measure the cache hit rate while varying the effective cache size by use of the "SET CACHE nnn" command. It is recommended that a similar procedure be carried out to determine the optimum cache size for a given application program.

The shared-file data caching facility is still available and should be used instead of the generalized data caching facility in the following cases:

a) If the primary goal is to speed up application programs which make heavy use of shared files, and the memory space which can be devoted to data caching is limited (less than 50 blocks), then the shared-file data caching facility is more effective than the generalized data caching facility.

b) If the size of the unmapped portion of the TSX-Plus system is such that 1800 bytes of code for the generalized data caching facility cannot be added. Note that the shared-file data caching

facility does not add any code to the unmapped portion of the system.

If the generalized data caching facility can be used, it is recommended that the shared-file caching facility not be used (it is redundant) and the NUMDC sysgen parameter be set to 0 (zero).

6. A virtual memory handler (VM.TSX) is now provided with TSX-Plus. It allows memory which is not allocated for use by the operating system to be used as a RAM based pseudo-disk device. VM is not supported as a bootable device. The VM handler uses the memory space above the top of memory used by TSX-Plus. TSX-Plus can be limited to using less than all installed memory by specifying the TSGEN MEMSIZ parameter. The MEMSIZ parameter accepts a numeric argument defining the number of K-bytes for TSX-Plus to use. For instance, the statement:

```
MEMSIZ = 256.
```

would restrict the TSX-Plus operating system to using only the first 256 K-bytes of memory. (Note: The decimal point is required, otherwise the number is interpreted as an octal value.)

A device definition entry for VM must be made in TSGEN. The correct VM device declaration is:

```
DEVDEF    <VM>,NONDMA
```

In order to use VM with TSX-Plus, the VM handler must be installed in RT-11 before running TSX-Plus. Users who do not have an RT-11 VM.SYS handler can create one by copying NL.SYS to VM.SYS. This VM.SYS, although not functional, may be installed in RT-11 so that TSX-Plus can load and use VM.TSX.

After TSX-Plus is started, VM must be initialized before it can be used. Since VM is implemented as a block structured device and each block contains 512 bytes, the number of blocks available to VM will be two times the number of K-bytes allocated. The directory does require some storage and therefore the number of blocks reported after initialization will be slightly smaller than this total. For instance, in a system which contains 512 K-bytes total physical memory and with MEMSIZ=256., VM will have 256 K-bytes available. After initialization, a directory of VM will then show slightly less than 512 blocks.

VM will normally calculate the correct base address to use to be just above the last address used by TSX-Plus. You may increase this base address. The format of the SET command used to adjust the base address used by VM is:

```
SET VM BASE=nnnnnn
```

where nnnnnn represents bits 6 through 22 of the base memory address (in octal) which VM is allowed to use. However, if you specify a base address below the top address of TSX-Plus, VM will dynamically adjust this base address back above the top of TSX-Plus. For example, if you wish to set the base address of VM to start after the first 512 K-bytes, then nnnnnn should be 20000 since the base memory address is 2000000 (octal). Any time a new base address is defined, VM should be initialized.

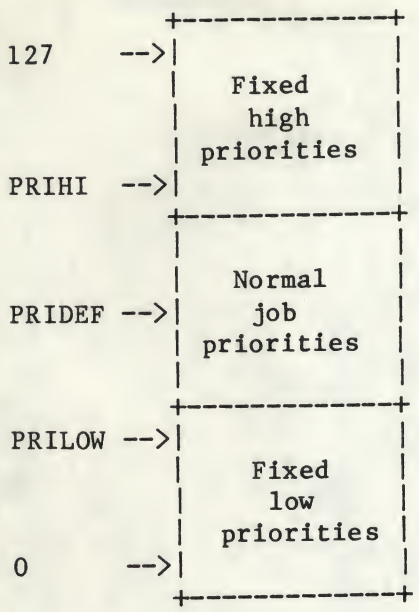
7. It is now possible to declare a command file that is to be executed when a job logs off. To declare a logoff command file, place a command of the following form in the startup command file for the job:

```
SET LOGOFF FILE=name
```

Where "name" is the file specification for the logoff command file. The SET LOGOFF command is only legal within the startup command file for a job. The logoff command file is executed whenever the job logs off. Be careful in what you place in a logoff command file since the execution of a logoff command file cannot be aborted by typing control-C.

8. A facility has been added to allow execution priorities to be specified for jobs. The execution priority values range from 0 to 127. Jobs with higher priority values take precedence over jobs which have lower priority values.

The priority values are arranged in three groups: the low priority group consists of priority values from 0 up to the value specified by the PRILOW sysgen parameter; the high priority group ranges from the value specified for the PRIHI sysgen parameter up to 127; the middle priority group ranges from (PRILOW+1) to (PRIHI-1). The following diagram illustrates the priority groups:



Jobs with priorities in the fixed-low-priority group (0 to PRILOW) and the fixed-high-priority group (PRIHI to 127) execute at fixed priority values. That is, the priority absolutely controls the scheduling of the job for execution relative to other jobs. A job with a unique fixed priority is allowed to execute as long as it wishes until a higher priority job becomes active. The fixed-high-priority group is intended for use by real-time programs. The fixed-low-priority group is intended for use by very low priority background tasks. Normal time-sharing jobs should not be assigned priorities in either of the fixed priority groups.

Jobs with the same priority in the fixed-high-priority group are scheduled in a round-robin fashion based on the QUAN0 sysgen parameter. If QUAN0 is set to 0 (zero), no round-robin scheduling is done for high-priority jobs. Jobs with the same priority in the fixed-low-priority group are scheduled in a round-robin fashion based on the QUAN3 sysgen parameter.

The middle group of priorities from (PRILOW+1) to (PRIHI-1) are intended to be used by normal, interactive, time-sharing jobs. Jobs with these assigned priorities are scheduled in a more sophisticated manner than the fixed-priority jobs. In addition to the assigned priority, external events such as terminal input completion, I/O completion, and timer quantum expiration play a role in determining the effective scheduling priority.

For most situations, the best strategy is to assign a single priority in the middle of the interactive job priority group to all interactive jobs and reserve the fixed priority groups for real-time or very low priority jobs. The default job priority is specified by the PRIDEF sysgen parameter.

The job priority may be specified by use of either a SET command or a system service call (EMT). The form of the set command is:

```
SET PRIORITY value
```

where "value" specifies the priority value for the job.

The form of the EMT is:

```
EMT    375
```

with R0 pointing to the following EMT argument block:

```
.BYTE  0,150
.WORD  value
```

where "value" is the priority value for the job.

The maximum priority value available to a user may be restricted when an account is authorized using the TSAUTH program. Existing accounts created with previous versions of TSX-Plus are assigned a maximum priority value of 50.

In addition to restricting the priority by use of the LOGON facility, there is also a keyboard command to restrict job priority. The form of this command is:

```
SET MAXPRIORITY value
```

Where "value" is in the range 0 to 127. The SET MAXPRIORITY command may only lower the maximum authorized priority value for the job, it may not increase it. Thus, the system manager may restrict job priority by placing a SET MAXPRIORITY command in the start-up command file for a line. This provides a method for controlling the maximum job priority for those lines that do not use the LOGON facility.

When a job is disassociated from a terminal by switching to a virtual line, the priority of the disassociated job is reduced by a value that may be specified as a system generation parameter -- PRIVIR. This automatic priority reduction does not apply to jobs in the fixed-low- or fixed-high-priority groups.

The following SHOW command can be used to display the current job priority, the maximum authorized execution priority, and the fixed-low- and fixed-high-priority ranges:

```
SHOW PRIORITY
```

The .GVAL EMT may be used to obtain the current and maximum authorized priority values for the job. The current priority is returned when a .GVAL is requested with an offset of -16 (decimal); the maximum

authorized priority is obtained by use of an offset of -18.; The value of the PRIHI sysgen parameter is obtained by use of an offset of -28. and the value of PRILOW is obtained by use of an offset of -26.

9. It is now possible to log terminal output to a file or device. When terminal logging is enabled, all output directed to the terminal is also written to the log file. The only exception to this is high-efficiency terminal output which is not logged. To initiate terminal logging, use the following command:

```
SET LOG FILE=name
```

where "name" is the device/file specification for the log file. The default extension is ".LOG". For example, the following SET LOG command opens a log file named "DK:RUNLST.LOG":

```
SET LOG FILE=RUNLST
```

The following SET command causes terminal output to be logged to the line printer:

```
SET LOG FILE=LP:
```

The following command closes the log file and terminates terminal logging:

```
SET LOG CLOSE
```

The log file is automatically closed when another log file is opened or the job logs off. The log file is also closed if the device containing the log file is initialized, squeezed, or dismounted.

The following SET commands can be used to suspend and resume terminal logging, they do not close the log file but merely control whether terminal output is written to the log file:

```
SET LOG NOWRITE
SET LOG WRITE
```

The following SET command resets the contents of the currently open log file, discarding everything that has been previously written to the log file and causing logging to start over at the front of the file:

```
SET LOG CLEAN
```

Terminal logging is especially useful with detached jobs. Normally all terminal output produced by detached jobs is discarded. However, the logging facility may be used to cause detached job terminal output to be directed to a file or printer.

10. Support is now provided for 18-bit DMA devices (such as DY) to be used with TSX-Plus on Q-bus systems which otherwise support 22-bit addressing (e.g., 11/23-PLUS and 11/73). It will NOT enable 22-bit addressing on machines in which the processor and backplane do not support 22-bit addressing.

Background: The original LSI Q-bus used with 11/23 systems had 18 address lines allowing I/O transfers to take place within 256Kb of memory. Device controllers developed during this period supported 18 address bits. With the introduction of the 11/23-PLUS processor, four additional address bits were added to the Q-bus bringing the total to 22 address bits which allowed I/O transfers to take place to 4Mb of memory. Unfortunately, many sites still have older device controllers that only support 18 bits and, in fact, DEC still does not build a Q-bus DY (RX02) controller that supports 22 bit DMA transfers. The 18-bit controllers will operate satisfactorily with 22-bit Q-bus systems provided that the I/O transfer is always within the lower 256Kb of memory. This has caused problems with TSX-Plus since jobs may be located anywhere in physical memory and I/O transfers are normally done directly to buffers located in the job region.

With version 5.0 of TSX-Plus, an option has been provided that will cause the system to "map" I/O transfers through system buffers that are always located in the lower 256Kb of memory. This facility may be specified selectively for those DMA devices that only have 18-bit controllers; devices which support 22-bit addressing do not need system buffering and can operate normally. When I/O mapping is selected for a device, TSX-Plus examines each I/O operation directed to the device and if the buffer is outside of the lower 256Kb it moves the data from the user's buffer to/from a system buffer and performs the actual data transfer from the system buffer to/from the I/O device. This allows 18-bit devices to be accessed by all time-sharing jobs regardless of their location in physical memory. However, it introduces a significant speed penalty since the data must be moved between the system buffer and the buffer in the job space. A further speed penalty is introduced in cases in which the amount of data being transferred is larger than the system buffer. In this case, an I/O operation which would normally be accomplished as a single transfer will be broken down into a series of smaller transfers. When a large operation is broken down into a series of smaller operations, time is lost waiting for the device to reposition itself for the start of the next operation. This speed penalty can be minimized by allocating a large enough system buffer to accommodate most I/O transfers as a single operation.

I/O mapping is invoked for a device by specifying "MAPIO" as the third parameter of the DEVDEF macro in TSGEN. Thus, the new form of the DEVDEF macro is:

```
DEVDEF <device>,[NON]DMA[,MAPIO]
```

For example, the following DEVDEF could be used to specify the device DY:

```
DEVDEF <DY>,DMA,MAPIO
```

Note that I/O mapping only needs to be specified for a device if ALL of the following conditions are met:

a) The system has more than 256Kb of memory.

b) The system has a Q-bus rather than a UNIBUS.

c) The device does direct memory access (DMA). Note that devices such as DX, LP, LS, and VM are not DMA.

d) The device controller or handler only supports 18-bit addressing. On the Q-Bus, the only device handlers which support 22-bit addressing (with 22-bit controllers) are DL, DU, and MS. DMA device handlers which only support 18-bit transfers and require I/O mapping if the above conditions are met are DM, DP, DS, DT, DY, MM, MT, RF, and RK.

Two TSGEN parameters are used to control the allocation of system buffers used for I/O mapping.

The MIOBSZ parameter specifies the size of the system I/O buffers. This parameter is specified in terms of the number of 512 byte areas to allocate for each buffer. The larger this parameter is, the faster will be the I/O transfers. The maximum value that may be specified is 15. Because directory operations occur in 1024 byte chunks, it is recommended that the MIOBSZ value not be less than 2. It is strongly suggested that the system disk have a controller that does not require I/O mapping; however, if the system disk does require I/O mapping, the MIOBSZ parameter should be set to 15.

The MIONBF parameter specifies the number of system I/O buffers to be allocated. These buffers are shared by all jobs (and the system itself) that are doing I/O to devices that require I/O mapping. Thus, if a system buffer is not available, the I/O transfer will be delayed until a buffer becomes available. The MIONBF parameter should be set to a value that corresponds to the number of devices which require I/O mapping and which will be simultaneously in use.

The system I/O buffers are allocated in the mapped portion of the system and are not part of the area that is constrained to 40Kb. No system I/O buffers are allocated (regardless of the value of MIONBF) if the system does not have a Q-bus, has no more than 256Kb, or if there are no devices that require I/O mapping.

11. A command has been added to control whether or not a terminal will accept messages transmitted by a SEND command from another terminal. The form of the command is:

```
SET TT [NO]GAG
```

If the SET TT GAG command is issued, messages sent by another terminal will be rejected unless the job receiving the message is running the keyboard monitor (KMON). The default mode is NOGAG.

12. A command has been added to allow the user to specify the prompt character(s) to be used by the keyboard monitor. The form of the command is:

```
SET PROMPT "string"
```

where "string" is a quoted character string from 1 to 8 characters long. For example, the following command could be used to set the default prompt to a dollar sign followed by a space (the VMS standard command prompt):

```
SET PROMPT "$ "
```

The following command would restore the standard RT-11 dot prompt:

```
SET PROMPT "."
```

13. The TIME command has been enhanced to accept seconds.

14. An EMT has been added to perform all of the terminal control functions that can also be performed using the "lead-in" character method. The form of the EMT is:

```
EMT    375
```

with R0 pointing to the following argument block:

```
.BYTE  0,152
.WORD  function_code
.WORD  argument_value
```

where "function_code" is a value that corresponds to the ASCII code for the letter that would be used with the lead-in character to perform the desired terminal control function. "Argument_value" is a value that is used with some functions and corresponds to the letter that would be sent following the function-code character if the lead-in sequence were used.

For example, to declare the slash character ("/") to be an activation character, you can either use the terminal control sequence: <lead-in>, "D", "/", or use an EMT 375 instruction with R0 pointing to the following argument block:

```
.BYTE  0,152
.WORD  'D
.WORD  '/
```

This EMT can be used to perform terminal control functions even when the terminal is in high-efficiency mode.

15. Two new TSX-Plus real-time EMT's have been added. The first allows an interrupt vector to be connected to a service routine in such a way as to allow much more rapid response than was possible using the other method of connecting interrupts to completion routines. The second EMT allows a real-time interrupt service routine to trigger the execution of a completion routine.

There are now two ways to connect real-time interrupts to TSX-Plus jobs. The original method used the mechanism of completion routines to perform the connection. When an interrupt occurred a completion routine request was queued for the job that was connected to the interrupt. This method was very general and allowed the completion routine to perform EMT's; also the job did not have to be locked in memory.

The new method provides a more direct connection between interrupts and service routines in TSX-Plus jobs. Rather than using the completion routine mechanism to call the service routine, the new method does a .INTEN call followed by a .FORK, sets up memory management for the job being called (but not full job context), and then enters the interrupt service routine. This approach minimizes the overhead in entering the service routine. Running on an 11/44, it appears that up to 2,000 interrupts per second can be processed.

There are more restrictions on a program using this method than the completion routine based method. Specifically, the following conditions must be met to directly connect to an interrupt:

a) The job must be locked in memory before connecting to the interrupt vector and must remain locked in memory as long as the interrupt connection is in effect.

b) The processing done by the service routine should not be very lengthy since other interrupts that do .FORKs will be queued up until the interrupt service routine exits.

c) Only two EMT's are legal within the interrupt service routine:

1) A .RSUM EMT which causes the program's main-line code to continue processing if it has done a .SPND EMT.

2) The EMT described below that schedules execution of a completion routine.

The interrupt service routine runs in user mode and uses the memory mapping that has been established for the job before the interrupt occurs. Access to the I/O page through PAR 7 is possible provided that this mapping has been set up in the main-line code before the interrupt occurs.

The interrupt service routine exits from the interrupt by executing a RETURN (RTS PC) instruction (not a RTI).

If an interrupt service routine needs to perform an EMT other than the two listed above (e.g., it needs to write a data buffer to a file) it should trigger a completion routine and have the completion routine perform the EMT for it.

The EMT to connect a service routine to an interrupt has the form:

```
EMT    375
```

with R0 pointing to the following EMT argument block:

```
.BYTE  20,140
.WORD  vector_address
.WORD  service_routine
.WORD  0
```

where "vector_address" is the address of the interrupt vector to which the service routine is to be connected and "service_routine" is the address of the entry point of the interrupt service routine.

If an error is detected by this EMT, the carry flag is set and one of the following error codes is returned:

Code	Meaning
1	There are no free interrupt control blocks
2	Some other job is connected to interrupt
3	Job is not locked in memory

The association between an interrupt service routine and an interrupt vector may be released by the same EMT as is used to release the association between an interrupt completion routine and an interrupt vector. That is:

```
EMT 375
```

where R0 points to the following argument block:

```
.BYTE   12,140
.WORD   vector_address
```

The EMT to trigger a completion routine has the form:

```
EMT    375
```

with R0 pointing to the following argument block:

```
.BYTE  21,140
.WORD  completion_routine
.WORD  priority
.WORD  R1_value
.WORD  0
```

Where "completion routine" is the address of the entry point of the completion routine that is to be started, "priority" is a value that indicates the execution priority for the completion routine, and "R1_value" is a value that is placed in register R1 when the completion routine is entered.

The completion routine execution priority is not the same as the hardware selected priority of the interrupt. All completion routines are synchronized with the job and run at hardware priority level 0. The completion routine priority is used by TSX-Plus to schedule the completion routine for execution. The larger the value of the specified priority, the higher the priority of the completion routine.

The execution of a real-time completion routine for one job will be interrupted and suspended if an interrupt occurs that causes a higher-priority completion routine for a different job to be queued for execution. A completion routine for a given job will never be interrupted to run another completion routine for the same job even if a higher-priority completion routine is pending. However, a completion routine can be interrupted by the execution of a directly connected interrupt service routine for the same job or any other job.

Completion routine priorities greater than zero are classified as "real time" priorities. They are higher than the execution priorities given to normal time-sharing jobs and will preempt the execution of any lower priority jobs.

The actual job priority used by the system for execution scheduling is computed by adding the completion routine priority to the PRIHI sysgen parameter which defines the base of the real-time priority range. The sum of the completion routine priority value and PRIHI is constrained so that it will not exceed the maximum allowed job priority of 127.

A completion routine running at a real-time priority is allowed to run continuously until one of the following events occurs: 1) the completion routine completes its execution and returns; 2) a higher priority completion routine from another job interrupts its execution -- it is re-entered when the higher priority routine exits; 3) the completion routine enters a system wait state such as waiting for an I/O operation to complete or waiting for a timed interval. If a completion routine enters a wait state, it returns to the same real-time priority when the wait condition completes.

If the QUAN0 sysgen parameter is greater than zero, completion routines that have the same execution priority are executed on a round-robin basis using the QUAN0 time-slice value. This time-slicing only applies to routines with the same priority. A routine with a higher real-time priority is never time-sliced with a routine with a lower priority.

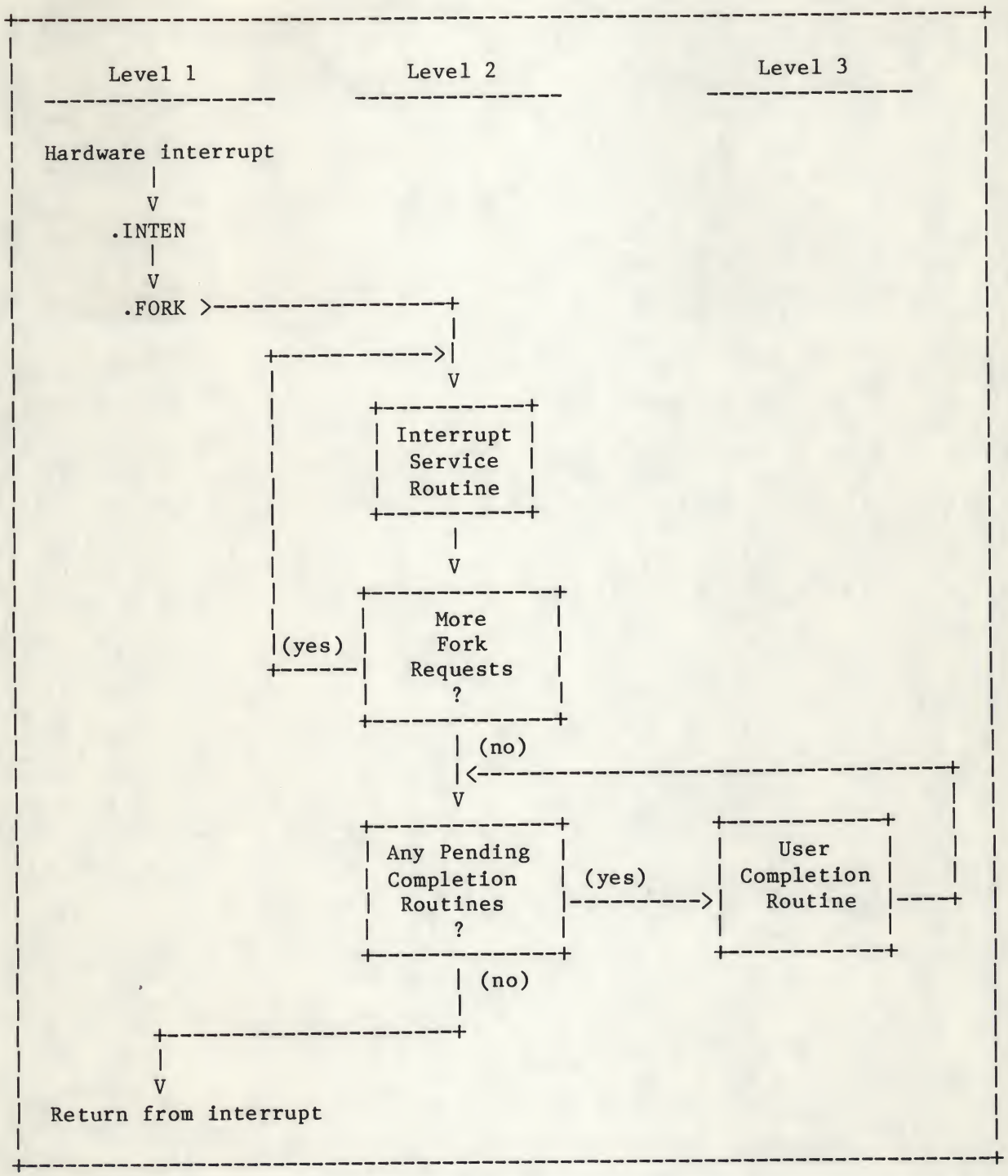
Note that a completion routine running at a real-time priority can indefinitely block the execution of all lower priority jobs and time-sharing users.

Completion routine priority 0 (zero) operates differently from the other priorities. Its action depends on whether the base priority of the job (as specified with the SET PRIORITY command or the corresponding EMT) is in the real-time priority range (greater than or equal to the PRIHI sysgen parameter). If the base priority of the job is greater than or equal to PRIHI, then priority 0 operates the same as the other real-time completion priorities and the completion routine executes with a scheduler priority equal to PRIHI (PRIHI+0). If the base priority of the job is less than PRIHI, then priority 0 is not classified as a "real time" priority but rather as a very high "normal" priority. The effect of this is that completion routines with priority 0 interrupt normal time-sharing jobs, but are time-sliced in the normal fashion and lose their high-priority if they execute longer than the length of time allotted by system parameter QUAN1A.

If additional requests are made to trigger the same or different completion routines while a completion routine is executing, the requests are queued for the job and are serviced in order based on their priority and the order in which they were queued.

The following diagram illustrates the processing of an interrupt:

Interrupt Processing



This diagram shows that there are three "levels" of interrupt processing. Level 1 is entered when a hardware interrupt occurs. In this level the processor (hardware) priority is set to 7 which causes other interrupt requests to be temporarily blocked. After some brief interrupt entry processing, the system performs a .FORK operation which queues up a request for processing at fork level and then drops the processor priority to 0. At this time another hardware interrupt can occur, in which case the cycle will be repeated and another request for fork-level processing will be placed on the queue.

Level 2 processing is also known as "fork level" processing. This level of interrupt processing services requests that were placed on a queue by the .FORK operation. Hardware interrupts are enabled during this processing and if any other interrupts occur their service requests are placed at the end of the fork request queue. Interrupt service requests are processed serially in the order that the interrupts occurred. Only a limited set of system service calls can be used from service routines running at fork level. One of the valid EMT's is a request to queue a user completion routine for subsequent processing.

Level 3 processing occurs in "job state". That is, the TSX-Plus job execution scheduler selects the highest priority job or completion routine and passes execution to it. Completion routines run with full job context and may issue system service calls as needed. Completion routines are serialized for each job. That is, all other completion routines (including higher priority interrupt completion routines) which are scheduled for the same job are queued for execution and will not be entered until the current completion routine exits. During level 3 processing, interrupts are enabled and job execution may be interrupted to process fork-level interrupt service routines or by higher priority completion routines for other jobs.

16. In order to improve interactive job performance, the job scheduler has been changed to give preference to interactive jobs. Each time a job accepts a character from the terminal (this is effectively the same as when a job receives an activation character), the job is classified as "interactive" and the following actions are taken:

a) The job is placed in the highest priority interactive execution state (other than real-time execution priorities).

b) A system timer is started for the job.

c) The I/O count for the job is set to zero.

The job remains in the highest priority interactive execution state until it either has executed for QUAN1C units of time or performs an I/O operation. At that time, the job is rescheduled into the next lower execution state (interactive-CPU). Jobs of equal priority in the

interactive-CPU state are scheduled on a "round-robin" basis for QUAN1B units of time each. Interactive jobs which accumulate a total of QUAN1 units of time are re-classified as non-interactive and placed in the CPU-bound state. On return from an I/O operation (during which the job was probably in an I/O wait state) an interactive job is placed in the interactive-CPU state. The job's I/O count is incremented for each I/O operation. If the number of I/O operations exceeds the value of the INTIOC parameter, then the job is reclassified as non-interactive. Non-interactive jobs are assigned lower execution priorities than interactive jobs.

These parameters which affect interactive job scheduling (QUAN1, QUAN1B, QUAN1C, and INTIOC) may be assigned initial values in TSGEN and may be changed while the system is running by use of the following keyboard commands:

```
SET QUAN1 value
SET QUAN1B value
SET QUAN1C value
SET INTIOC value
```

The QUAN1, QUAN1B, and QUAN1C parameters are specified in units of 0.1 seconds.

In selecting values for these parameters, the following guidelines should be considered: It is highly desirable that interactive jobs such as data entry applications and editing programs be classified as "interactive" throughout each terminal interaction. Thus, QUAN1 should be set large enough so that the total CPU time used by the application program during one interaction can be completed. Note that if a job performs I/O operations, the CPU counter is suspended (time is not charged while a job is in a wait state) and restarted (but not re-initialized) when the I/O operation completes. Also, the INTIOC parameter should be set to a value large enough to allow all I/O operations required during a single interactive transaction to be completed.

It is much better to select values for QUAN1 and INTIOC that are too large rather than too small. If the values are too large they will allow long running (non-interactive) programs to be scheduled as interactive slightly longer than necessary. If they are too small, interactive jobs will be reclassified as non-interactive (and given a lower priority) while they are executing an interactive transaction.

The QUAN1B and QUAN1C parameters should be set to a small value which is just sufficient to provide enough CPU time for brief interactive operations such as character processing by interactive editors such as KED. The recommended value is in the range 1 to 4. See the description of the SET SIGNAL command below for a technique to determine appropriate values for these parameters.

A switch is available for use with the RUN and R commands to execute a program without giving the program the benefit of the interactive-job priority boost facility. This should be used with programs that do heavy terminal I/O but which are not really interactive -- such as file transfer programs. The command to execute a program non-interactively is:

```
R[UN]/NONINTERACTIVE program
```

Where the switch name may be abbreviated to "/N".

The effect of this switch is to cause the program to be classified as non-interactive even though it does terminal I/O. The result is that the program will run at a lower priority and will not interfere with interactive programs. The effect of this switch is cleared when a program exits to KMON (but not if the program does a .CHAIN).

An EMT is also available to set the interactive/noninteractive nature of a program. The form of this EMT is:

```
EMT    375
```

with R0 pointing to the following argument block:

```
.BYTE  0,153
.WORD  value
.WORD  0
```

If "value" is 1 the job is declared to be interactive and if value is 0 the job is declared to be non-interactive just as if the /NONINTERACTIVE switch had been used with the RUN command.

17. A system generation parameter has been added to allow the system manager to control the execution priority boost that is given to Input/Output active jobs. Normally each time a job is restarted upon I/O completion the job is given an execution priority boost and remains in a high-priority execution state until either (1) it goes into a wait state such as waiting for another I/O operation; or (2) it has executed for QUAN1A units of time at which point it is rescheduled in the CPU-bound execution state.

A new system generation parameter named HIPRCT controls the number of consecutive times a job will be given a priority boost due to I/O completion between times when the job goes into a CPU-bound state or a terminal-input bound state. After a job has been given HIPRCT priority boosts, the job is scheduled in the CPU-bound execution state and the boost count is reinitialized.

If HIPRCT is given a large value (say 1000), the operation of the system will be the same as it was in earlier versions of TSX-Plus which gave an unlimited number of priority boosts to jobs. If the value is set to 0 (zero), jobs will always be placed in the CPU-bound queue when I/O completion occurs. The recommended value for HIPRCT is somewhere in the range of 5 to 50. The higher the value, the better the performance of I/O intensive jobs but the more they will tend to dominate the system.

A keyboard SET command may be used to dynamically alter the value of the HIPRCT parameter. The form of the SET command is

```
SET HIPRCT value
```

18. A "SET SIGNAL" command has been added to the system to aid the system manager in selecting system tuning parameters. The form of the SET SIGNAL command is:

```
SET SIGNAL [NO]parameter
```

Where "parameter" is one of the following system parameters: QUAN0, QUAN1, QUAN1A, QUAN1B, QUAN1C, QUAN2, QUAN3, INTIOC, or HIPRCT.

When signaling has been set for a system parameter, the bell will be rung at the terminal of the job which set the signal each time a job state transition occurs because the job has reached the specified parameter value. This allows the system manager to observe how often the job changes state based on different parameter values.

The SET SIGNAL command operates on a line-by-line basis and affects only the line that issued the command.

Signaling may be turned on for any combination of parameters, but each parameter must be specified by a separate SET SIGNAL command. Signaling for an individual parameter may be turned off by specifying "NO" in front of the parameter name. All parameter signaling may be turned off by use of the following command:

```
SET SIGNAL OFF
```

The signaling feature is intended to provide an aid to the system manager in determining optimum job scheduling parameters for the particular variety of jobs at each unique TSX-Plus installation. For example, jobs are classified as "interactive" from the time that a character is received from the terminal until QUAN1 units of CPU time are used or INTIOC I/O operations have been performed. Thus, if the following SET SIGNAL commands are entered:

```
SET SIGNAL QUAN1
SET SIGNAL INTIOC
```

then the bell will ring each time the job switches from interactive to non-interactive state.

The recommended procedure for selecting values for QUAN1 and INTIOC is as follows:

a) Issue the following keyboard commands:

```
SET SIGNAL QUAN1
SET SIGNAL INTIOC
```

b) Set INTIOC to a large value by use of the following keyboard command:

```
SET INTIOC 1000
```

c) Run an application program whose execution is to be optimized.

d) From a separate terminal vary the value of QUAN1 by use of the keyboard set command:

```
SET QUAN1 value
```

For each trial value of QUAN1 enter several transactions to the application program and see if the bell rings at the terminal running the application program. If the bell rings, increase the value of QUAN1 and try again. The optimum value of QUAN1 is slightly larger (add 1 to 5) than the smallest value found which is large enough so that the bell does not ring while processing a transaction.

e) Repeat the process for INTIOC by setting QUAN1 to a large value (e.g., 1000) and vary INTIOC starting with a reasonable value such as 30.

f) Try several values of INTIOC until the smallest value is found which is large enough to keep the bell from ringing while processing a single transaction. The optimum value for INTIOC is slightly larger than this (i.e., add 2 to 10).

g) After the appropriate value for QUAN1 and INTIOC have been determined, the system default values for these parameters may be set by modifying TSGEN and regenerating TSX-Plus.

Note: When performing this type of optimization, choose the most frequent and important type of transactions for the test. Don't worry about longer and less frequent operations such as chaining between separate programs. Also note that system parameters such as QUAN1 and INTIOC (as well as QUAN1A, QUAN1B, etc.) are global to all users and may not be set on a line-by-line basis.

19. The SHOW command has been enhanced to allow it to display values for the following system parameters: QUAN0, QUAN1, QUAN1A, QUAN1B, QUAN1C, QUAN2, QUAN3, CORTIM, INTIOC, HIPRCT, CACHE, NUMDC, PRIHI, PRILOW, PRIVIR, and PRIDEF. The form of the SHOW command is:

```
SHOW parameter
```

20. The SHOW DEVICES command has been enhanced to provide more information about the devices and their associated handlers. A typical SHOW DEVICES display is shown below:

Device	Status	Handler base	Handler size	CSR	Vector
TT	000004				
DB	100021	066416	448	176700	254
MT	016011	067746	3844	172520	224
DX	102022	077352	594	177170	264
LP	020003	100474	318	177514	200
NL	000025	101172	58		
LD	102446				

The addresses of the CSR and vectors for the devices are not displayed for system pseudo devices such as TT and LD, nor are they displayed if the person issuing the SHOW DEVICES command does not have operator privilege.

21. An EMT has been added to control whether a spooled file is held until it is closed or released to the spooled device as soon as anything has been written to the file. The effect of this EMT is similar to the SPOOL LP,[NO]HOLD command, but on an individual file basis. The form of the EMT is:

```
EMT    375
```

with R0 pointing to the following EMT argument block:

```
.BYTE  chan,151
.WORD  0
.WORD  flag
```

where "chan" is the number of the channel that is open to the file, and "flag" is an indication of whether the file is to be held or released immediately. If the value for "flag" is 1, the file is held; if the value is 0, the file is released.

This EMT must be issued after the spooled file is opened but before any data is written to the file. If the channel is opened to a disk file or other non-spooled device, the EMT is ignored.

22. The directory caching facility has been enhanced in the following ways:

a) Directories for logical disks are now cached.

b) The system keeps track of which jobs have which disks mounted.

c) SQUEEZE and INITIALIZE commands will be rejected by the system if any other jobs have the disk mounted.

d) The directory cache is flushed less frequently than it was in previous versions of the system.

As with previous versions of the system, the MOUNT command is used to turn on directory caching for a device. There are two forms of the MOUNT command, one form simply enables directory caching; the other form enables directory caching and mounts a logical subset disk. The forms of these commands are:

```
MOUNT ddu
MOUNT LDn filspc [logical-name]
```

The DISMOUNT command is used to dismount logical subset disks and to turn off directory caching. The following information message is printed when a disk is dismounted and other jobs still have the disk mounted:

```
?KMON-I-Device is still mounted by other users
```

The system disk (SY:) is automatically mounted for each job as the job logs on. If any user has a disk mounted, all users who access the disk benefit from directory caching.

The system now keeps track of which jobs are associated with each disk by virtue of having mounted the disk. The SHOW MOUNTS command can be used to display a list of all mounted disks and the jobs which have the disks mounted. An example of a SHOW MOUNTS command display is shown below:

```
  Device          Associated jobs
---------- ----------------------------
DB0:         1  3  9
DB1:         1
DB6:         3
DB6:INSGDE   3
```

In this example, jobs 1, 3, and 9 have disk DB0 mounted; DB1 is mounted by job 1; DB6 is mounted by job 3; a logical disk named INSGDE on DB6 is also mounted by job 3.

When an INITIALIZE or SQUEEZE command is issued, the system checks to see if the device being initialized or squeezed is mounted by any jobs

other than the one issuing the command. If so, the command is rejected with the error message:

?KMON-F-Device is mounted by another user

Note that the system does not prevent jobs from accessing or writing to disks which are not mounted by that job. Extreme caution is still necessary when squeezing disks since severe data corruption is possible if a disk is squeezed while another job is writing to that disk without having mounted it.

23. In cooperation with Digital Equipment Corporation, a bit has been allocated in the RT-11 sysgen options word at monitor offset 372 to indicate whether a program is running under RT-11 or TSX-Plus. The TSX-Plus flag is the high order bit (mask 100000) of the sysgen options word. This bit will be set (1) when a program is running under TSX-Plus and reset (0) when running under any version of RT-11.

24. Several new values may now be obtained by use of the .GVAL EMT:

Offset	Value
-16.	Current execution priority
-18.	Maximum authorized priority
-20.	Number of blocks per job in SY:TSXUCL.TSX file
-22.	Job # of primary line (0 if this is primary line)
-24.	Name of physical SY: device
-26.	Value of PRILOW sysgen parameter
-28.	Value of PRIHI sysgen parameter

25. Approximately 6Kb of code and data have been moved from the unmapped, low memory portion of TSX-Plus into the mapped area. This provides additional space for device handlers or additional time-sharing lines. The exact amount of additional low memory gained will depend on which optional features are included in your current TSGEN.

26. The VTCOM/TRANSF file transfer programs may be used to communicate and transfer files between RT-11 and TSX-Plus systems or between two TSX-Plus systems.

When VTCOM is used to communicate with another system, the system where the user is located and running VTCOM is known as the "local" system whereas the remote system to which communication is taking place is known as the "host" system. TSX-Plus may be used either as the local system, the host system, or both.

The user at the local system runs the VTCOM program to initiate communication with the host system. The VTCOM program uses the XL handler (or XC on the PRO-350) to connect to a communications line. The XL handler must be set up to drive a DL11 or DLV11 communications port that is connected either directly or through a modem to the host system.

When TSX-Plus is used as the local system, a modified version of the XL handler (supplied with this release) must be used. XL must be specified as a device to TSX-Plus by use of a standard TSX-Plus device definition in TSGEN of the form

```
DEVDEF <XL>,NONDMA
```

An XL handler must also be installed in RT-11. If VTCOM is to be used only with TSX-Plus, the NL (null) handler can be copied to a file named XL.SYS and this dummy handler installed in RT-11.

The address of the interrupt vector and CSR register for the DL11 line to be controlled by XL must be set before XL can be used with the system. This can be done using the following commands which must be executed under RT-11 before TSX-Plus is started:

```
RENAME/SYS XL.SYS XL.TMP
RENAME/SYS XL.TSX XL.SYS
SET XL VECTOR=xxx
SET XL CSR=xxxxxx
RENAME/SYS XL.SYS XL.TSX
RENAME/SYS XL.TMP XL.SYS
```

When TSX-Plus is used as the local system, the IOABT sysgen parameter must be set to 1 to enable handler abort entry code.

The VTCOM program references the XL device by use of the logical name XC. So an assign of the following form must be used before running VTCOM:

```
ASSIGN XL XC
```

When TSX-Plus is used as the host system, the connection from the local system may be made through any TSX-Plus time-sharing line on the host system. The TRANSF program must be available on the host system if file transfers are to be performed.

27. System support for logical subset disk directories (LD) has been made a sysgen option. If the sysgen parameter LDSYS in TSGEN is set to 0 (zero), system support for logical disks is excluded. This may be done to allow a user written LD device handler to be used rather than the standard LD system support.

28. You may now type ahead during start-up command file processing.

29. The SYSMON program can now be started from within a command file.

30. The start-up command file initiation command line is no longer displayed during logon. If you put the two character sequence "^(" at the front of your start-up command files, you can completely hide the execution of the start-up command files.

31. The logon greeting message for virtual lines has been shortened.

32. An error message is now printed if an attempt is made to SQUEEZE, INITIALIZE, or FORMAT the system disk.

33. A "/VM" switch is now accepted for the LINK and EXECUTE CCL commands. It causes a COBOL-Plus program being linked to use the VM handler for locating the COBOL-Plus run-time file and for allocating the data segment swap file during execution. The use of this switch requires COBOL-Plus version 5.0.

Corrected Problems

34. The following problems with previous versions of TSX-Plus have been corrected:

34.1 A problem has been corrected which caused the system to hang or crash if a command were issued which attempted to execute a command file on TT: (such as "@TT:file" or "TT:file").

34.2 A problem has been corrected which caused characters typed ahead during KED exit processing not to be echoed.

34.3 A problem has been corrected which under some conditions caused handlers that did internal queuing to fail.

34.4 A problem has been corrected which in some cases caused typed-ahead characters not to be translated to upper case.

34.5 A problem has been corrected which prevented job error severity level codes (passed to KMON in location 53) from being passed on to the IND indirect command file processor.

34.6 A problem has been corrected which prevented FILEX from running under TSX-Plus.

34.7 A problem has been corrected in the expansion of the EDIT/TECO/EXECUTE:file command which caused it to improperly handle the first line of input.

34.8 Three CCL commands expansions have been adjusted. The LIBRARY/LIST command does not produce a library object file unless the /OBJECT switch is specified. The LINK/SYMBOLTABLE no longer requires an argument. The name of the symbol table file is determined by the switch position. The DELETE/INTERCHANGE command was altered to accept either the /QUERY or /NOQUERY switch.

34.9 A problem has been corrected with the .RENAME EMT which previously allowed a user to rename a file even though the user did not have write access to the file.

34.10 The .FETCH EMT now returns an error code of 0 if an attempt is made to fetch a handler for a device that is not installed in the system. Previously the .FETCH would return without an error status.

34.11 Handlers which used the RT-11 documented procedure for entering system state (by adjusting the stack to simulate an interrupt and issuing a .INTEN) would correctly set the C-bit upon return if an error was reported. A problem has been resolved so that the user's EMT error byte is now valid when using this method.

34.12 Two corrections have been made for I/O error handling. The .WRITW and .WAIT requests now clear the channel status hard error bit following the setting of the user's EMT error byte. The .READ, .READC, .WRITE, and .WRITC requests now analyze the channel status word and report errors previously encountered.

34.13 A correction was made to the P1EXT routine which fixed a problem for handlers which called this external system routine. The problem caused the DX driver not to zero fill the sector on a short write.

34.14 The C.USED word contained in the channel status information area is now cleared when a non-file structured .ENTER is issued, and the highest block number is retained when executing writes.

34.15 Three handlers have been corrected. The [NO]WRITE set option for both the DX and DY handler has been disabled and is unsupported under TSX-Plus. The RT-11 version 5.0 DD handler has been corrected to allow higher transfer rates under TSX-Plus. Baud rates as high as 9600 have been successfully achieved on an LSI-11/23-PLUS. This may vary depending on your hardware configuration and system load conditions.

34.16 In previous versions, the SEND command would terminate a user name at the first blank. This has been corrected to truncate only trailing spaces, thus reporting the entire name.

34.17 A problem has been corrected in the terminal handler which sometimes misdirected characters to the wrong port on some non-DEC DZ type interface cards.

* Software Problem Report *

From:	Mail to: S&H Computer Systems, Inc. 1027 17th. Avenue South Nashville, Tennessee USA 37212
Contact:	
Phone:	Phone: (615)-327-3670
Distributor:	Telex: 786577 S AND H UD
Date:	

S&H product: TSX:___ TSX-Plus:___ COBOL-Plus:___ RTSORT:___
INDAS:___ Other:_____

Version #:_____ License #:_____ CPU Model:_____

Report Classification

___ Severe error	___ Suggestion
___ Moderate error	___ Inquiry
___ Minor error	___ Comment
___ Documentation error	___ Other

Problem description / Comment:

S&H Response: Received:_____ Verified:_____ Resolved:_____

* Software Problem Report *

From:

Contact:

Phone:

Distributor:

Date:

Mail to: S&H Computer Systems, Inc.
1027 17th. Avenue South
Nashville, Tennessee USA
37212

Phone: (615)-327-3670

Telex: 786577 S AND H UD

S&H product: TSX:___ TSX-Plus:___ COBOL-Plus:___ RTSORT:___
INDAS:___ Other:_____

Version #:_____ License #:_____ CPU Model:_____

Report Classification

___ Severe error
___ Moderate error
___ Minor error
___ Documentation error

___ Suggestion
___ Inquiry
___ Comment
___ Other

Problem description / Comment:

All submissions become the property of S&H Computer Systems, Inc.

S&H Response: Received:_____ Verified:_____ Resolved:_____